Journal of Global Optimization (2021) 79:521–566
https://doi.org/10.1007/s10898-020-00952-6

# The DIRECT algorithm: 25 years Later

**Donald R. Jones**[1] · **Joaquim R. R. A. Martins**[1]

Received: 13 September 2019 / Accepted: 20 September 2020 / Published online: 17 October 2020


**Abstract**
Introduced in 1993, the DIRECT global optimization algorithm provided a fresh approach to minimizing a black-box function subject to lower and upper bounds on the variables. In contrast to the plethora of nature-inspired heuristics, DIRECT was *deterministic* and had only one hyperparameter (the desired accuracy). Moreover, the algorithm was simple, easy to implement, and usually performed well on low-dimensional problems (up to six variables). Most importantly, DIRECT balanced local and global search (exploitation vs. exploration) in a unique way: in each iteration, several points were sampled, some for global and some for local search. This approach eliminated the need for "tuning parameters" that set the balance between local and global search. However, the very same features that made DIRECT simple and conceptually attractive also created weaknesses. For example, it was commonly observed that, while DIRECT is often fast to find the basin of the global optimum, it can be slow to fine-tune the solution to high accuracy. In this paper, we identify several such weaknesses and survey the work of various researchers to extend DIRECT so that it performs better. All of the extensions show substantial improvement over DIRECT on various test functions. An outstanding challenge is to improve performance *robustly* across problems of different degrees of difficulty, ranging from simple (unimodal, few variables) to very hard (multimodal, sharply peaked, many variables). Opportunities for further improvement may lie in combining the best features of the different extensions.

**Keywords** DIRECT · Global optimization · Lipschitzian optimization · Black-box · Derivative-free · Exploitation versus exploration

## 1 Introduction

When it was published a little over 25 years ago, the DIRECT global optimization algorithm offered a new approach to minimizing a black-box objective function subject to lower and upper bounds on the variables [30]. Unlike genetic algorithms, simulated annealing, and other global optimization heuristics inspired by nature, DIRECT was motivated by Lipschitzian global optimization [22,54,55,67]—a rigorous and deterministic optimization method based on branch-and-bound, where bounds are computed based on knowledge of a Lipschitz con-

✉ Donald R. Jones
donjon@umich.edu

[1] Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI, USA

stant for the objective function (an upper bound on the rate of change). DIRECT introduced modifications to the Lipschitzian approach that made it more tractable in higher dimensions and eliminated the need to know the Lipschitz constant. The result was an algorithm that was truly black-box, simple, deterministic, and had only one hyperparameter (the desired accuracy). On the low-dimensional (2 to 6 variable) test problems used in the original paper, DIRECT performed extremely well compared to other methods.

DIRECT has also proven effective on low-dimensional, real-world problems from a wide range of fields, including genetics [42], surgery [76], air transport [8], power generation [2,13,34], astronomy [4], hydrogen fuel cell management [74], photovoltaic systems [49], hard disk design [77], aircraft routing [1], and many others [3,6,20,24,35,44,58,71]. However, DIRECT has been found to be less competitive on higher-dimensional problems [47]. Moreover, many authors have noticed the tendency for DIRECT to get close to the global optimum quickly, but take a long time to refine the solution to high accuracy [38,39,51,63].

To address these weaknesses, several researchers have offered variations to the original DIRECT algorithm, some of which have been successful. For example, in a recent comparison of derivative-free algorithms on a large set of test problems with up to 300 variables, a variation of DIRECT (glcCluster) was among the top performers [57]. Another promising variation, called eDIRECT-C [36,37], was recently successfully applied to optimizing a quantum simulator with respect to 15 parameters [31].

In this paper, we review some of the more important variations of DIRECT that have appeared in the literature. We begin in Sect. 2 with a recap of the original DIRECT algorithm, summarizing its strengths and weaknesses. In Sect. 3, we survey several published modifications of DIRECT, indicating how each of them addresses the identified weaknesses. In the conclusion, we categorize the key innovations that have been introduced and discuss opportunities for further research.

## 2 The original DIRECT algorithm

### 2.1 Algorithm description

The original DIRECT algorithm minimized a black-box objective function subject to lower and upper bounds on the design variables [30]; that is, it solved the problem:

$$\begin{aligned} \underset{x_1,\ldots,x_n}{\text{minimize}} \quad & f(x_1,\ldots,x_n) \\ \text{subject to} \quad & \ell_k \le x_k \le u_k \quad k = 1,\ldots,n. \end{aligned}$$

Several years later, the same authors published a revision of DIRECT that added the capability to handle nonlinear inequality constraints and integer variables, as well as a few other changes to speed up convergence [29]. Even though the revised version is arguably better, we will start by describing the original version since it has been the point of departure for many, if not most, of the subsequent extensions. We will cover the revision in a later section as part of our general review of enhancements to DIRECT.

Because DIRECT assumes lower and upper bounds we can, without loss of generality, normalize the design variables to [0, 1] so that the search space becomes the unit hypercube. DIRECT works by partitioning the unit hypercube into subrectangles with the property that the objective function has been evaluated at each rectangle's center point. In each iteration, certain "potentially optimal" rectangles are selected for further search; these rectangles are then subdivided and the function is evaluated at the center points of the newly-formed sub-

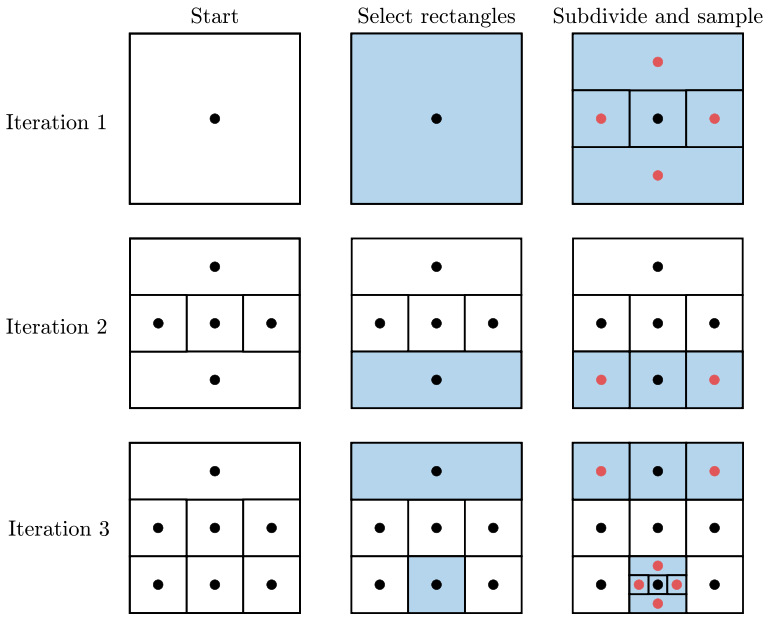


**Fig. 1** First three iterations of the DIRECT algorithm for a two-variable test function

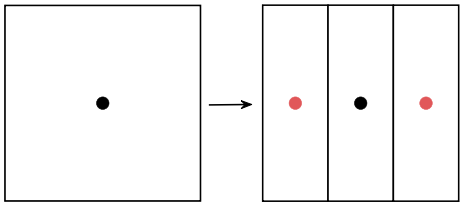

**Fig. 2** Trisecting the unit square along the first dimension

rectangles. Figure 1 illustrates this process, showing the first three iterations of DIRECT for a two-variable test function. At first, there is only one rectangle (the entire unit hypercube), so we select it. This rectangle is then subdivided into several subrectangles, and the center points of the newly formed rectangles (red dots) are evaluated. The center point of the original rectangle becomes the center point of one of the smaller rectangles formed after subdivision. In iteration 2, just one rectangle is selected, subdivided, and sampled. In iteration 3, we select two rectangles, and then sample and subdivide them. The process continues until we reach a prespecified limit on the number of iterations or function evaluations.

Clearly, the key elements of the DIRECT algorithm are the method for selecting "potentially optimal" rectangles and the method for sampling and subdividing these rectangles.

A simple method for sampling and subdividing a rectangle, illustrated in Fig. 2, would be to select one dimension arbitrarily and split the hypercube into thirds along this dimension. The original center point becomes the center of the middle third, so we only need to sample two new points (the centers of the newly created "left" and "right" thirds, shown as red dots).

However, arbitrarily selecting a dimension for trisection is not attractive. To avoid such arbitrariness, the original DIRECT algorithm uses the approach shown in Fig. 3. We start by sampling the points $\mathbf{c} \pm \delta \mathbf{e}_k$, $k = 1, \ldots, n$, where $\mathbf{c}$ is the center point of the hypercube, $\delta$ is one-third the side length of the hypercube, and $\mathbf{e}_k$ is the $k$th unit vector. In Fig. 3,

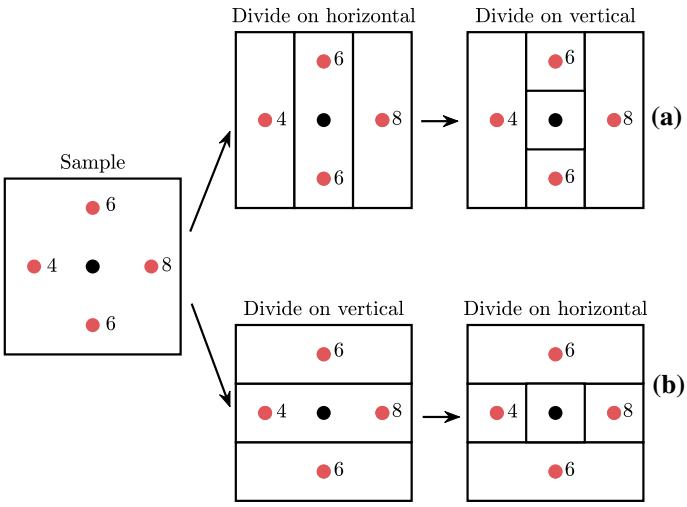


**Fig. 3** Sampling and dividing a square in the DIRECT algorithm

this translates into sampling above, below, to the left, and to the right of the center point; these newly sampled points are shown as red dots with numbers beside them indicating the function's value. By sampling along all dimensions, we have avoided selecting any dimension arbitrarily. But we must now resolve another issue: How do we divide the hypercube so that each subrectangle has a sampled point at its center?

Figure 3 shows two possible ways to do this for $n = 2$. In Fig. 3a, we first divide the square into thirds along the horizontal dimension and then divide the center rectangle (the one with $\mathbf{c}$) into thirds along the vertical dimension. In Fig. 3b, the order is reversed. Both of these division strategies partition the hypercube into subrectangles with sampled points at their centers. To decide which division order to use, notice that, if we first split on dimension $k$, then the two points $\mathbf{c} \pm \delta \mathbf{e}_k$ will be at the centers of the biggest subrectangles. For example, in Fig. 3a, we first divide in dimension 1; as a result, the points with function values 4 and 8 become the centers of the largest subrectangles. This observation leads to the following question: Do we want the biggest rectangles to contain the points with the best or the worst function values? The strategy used by DIRECT is to make the biggest rectangles contain the best function values. This strategy increases the attractiveness of searching near points with good function values (as we explain later, bigger rectangles are preferred for sampling, everything else equal). In our experience, the increased emphasis on local search speeds up convergence without sacrificing the global properties of the algorithm, which are ensured by the method of selecting rectangles discussed later.

More precisely, DIRECT adopts the following rule for subdividing a hypercube. Let

$$w_k = \min\{f(\mathbf{c} + \delta \mathbf{e}_k),\ f(\mathbf{c} - \delta \mathbf{e}_k)\} \tag{1}$$

be the best of the function values sampled along dimension $k$. We start by splitting along the dimension with the smallest $w$ value. Once this is done, we split the rectangle containing $\mathbf{c}$ into thirds along the dimension with the next smallest $w$ value. We continue in this way until we have split on all dimensions. This splitting rule would select Fig. 3a.

Once the initial hypercube has been divided, some of the subregions will be rectangular. In dividing such rectangles, we only consider the long dimensions. For example, the three-

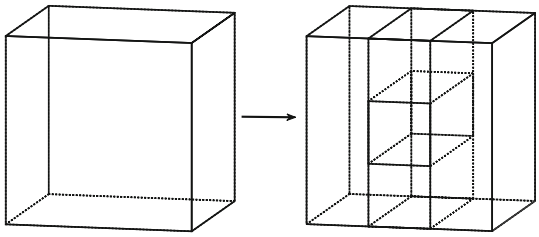

**Fig. 4** Dividing a hyperrectangle in the DIRECT algorithm

dimensional rectangle shown in Fig. 4 would be divided along the horizontal and vertical dimensions, but not the shorter depth dimension. By dividing only along the long dimensions, we ensure that the rectangles shrink in every dimension. The rectangle division procedure is summarized in Algorithm 1.

**Algorithm 1** Procedure for dividing rectangles

1: Identify the set $M$ of dimensions with the maximum side length. Let $\delta$ equal one-third of this maximum side length.
2: Sample the function at the points $\mathbf{c} \pm \delta \mathbf{e}_k$ for all $k \in M$, where $\mathbf{c}$ is the center of the rectangle and $\mathbf{e}_k$ is the $k$th unit vector.
3: Divide the rectangle containing $\mathbf{c}$ is into thirds along the dimensions in $M$, starting with the dimension with the lowest value of $w_k = \min\{f(\mathbf{c} + \delta \mathbf{e}_k), f(\mathbf{c} - \delta \mathbf{e}_k)\}$, and continuing to the dimension with the highest $w_k$.

Let us now turn our attention to how DIRECT selects the "potentially optimal" rectangles. As we mentioned earlier, DIRECT was motivated by Lipschitzian optimization so, just for the moment, let us assume that we knew a Lipschitz constant for the objective function, that is, we knew a constant $K$ such that, for any two points $\mathbf{x}$ and $\mathbf{x}'$ we have

$$|f(\mathbf{x}) - f(\mathbf{x}')| \leq K \|\mathbf{x} - \mathbf{x}'\|. \tag{2}$$

In Eq. (2), the norm on the right hand side can be the standard Euclidean 2-norm or other non-Euclidean norms. The original algorithm and most extensions of it use the 2-norm, but later we will discuss a variation of DIRECT called DIRECT-l that uses the infinity norm.

Now let $\mathbf{c}_i$ be the center of rectangle $i$ in the partition, and let $d_i$ be the Euclidean distance between $\mathbf{c}_i$ and the vertices of rectangle $i$ (see Fig. 5). Given that the function has a value $f(\mathbf{c}_i)$ at the center of rectangle $i$ and that the maximum distance between $\mathbf{c}_i$ and any point in rectangle $i$ is $d_i$, it follows that a valid lower bound for $f$ over rectangle $i$ is:

$$\text{lower bound for } f \text{ over rectangle } i = f(\mathbf{c}_i) - K d_i.$$

In standard Lipschitzian optimization, in each iteration we would compute such a lower bound for every rectangle in the partition and select the rectangle with the lowest lower bound for subdivision and further sampling. If $K$ is a valid Lipschitz constant (upper bound on the rate of change of the objective function), then one can prove that, for any small positive number $\epsilon > 0$, the algorithm will find a solution within $\epsilon$ of the optimum in a finite number of iterations.

If we look at formula for the lower bound, we see that it is a combination of two terms. The first term, $f(\mathbf{c}_i)$, is lower (and therefore better) when the function value at the rectangle center is low. Thus, this term leads us to select hyperrectangles with good function values,

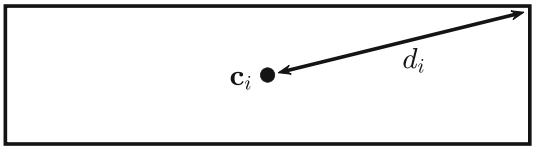


**Fig. 5** The center-vertex distance $d_i$ of rectangle $i$

that is, it leads us to do local search. The second term is lower algebraically the bigger the hyperrectangle (larger $d_i$). Thus, this term leads us to select hyperrectangles with large amount of unexplored territory, that is, it leads us to do global search. The Lipschitz constant $K$ serves as a relative weight on global versus local search; the larger $K$, the higher the relative emphasis put on global search. However, since the Lipschitz constant must be an upper bound on the rate of change of the objective function, it is generally quite high. In terms of the above discussion, this means that a standard Lipschitzian approach places a high emphasis on global search and, as a result, it may exhibit slow convergence.

Once the basin of the optimum is found, the search would proceed more quickly if $K$ could be reduced, thereby putting more emphasis on local search. In fact, several authors have proposed heuristic ways to vary the Lipschitz constant across the design space [23], using a lower Lipschitz constant in regions where the objective function seems "flat" based on the sampled points in that region. Sergeyev et al. [32,60,61,64] developed a sophisticated technique for determining the Lipschitz constant to use in a particular region; in particular, they use a weighted combination of local and global estimates of the Lipschitz constant, automatically reducing the weight on the local estimate in sparsely sampled regions where the local estimate of the Lipschitz constant may be poor. An important feature of this local tuning approach is that the local (exploitation) and global (exploration) information is balanced automatically without user intervention. These methods often perform well and ensure the convergence to the global minimizers only, under rather general conditions (see, for example, [64]). However, such methods (see also [70]) require the user to specify a "reliability parameter" $r > 1$ with the property that $r$ times the highest observed rate of change between sampled points is a valid Lipschitz constant. In general, the performance of the methods is sensitive to the choice of $r$: a value that is too high makes the algorithm slow to converge, while a value that is too low may cause the algorithm to not converge to the optimum. Even so, a correct estimate of $r$ can often be obtained either theoretically (from the convergence analysis of a global optimization method with local tuning) or from the analysis of properties of a practical decision making problem to be solved [64,70].

DIRECT avoids the drawback of using a single, large Lipschitz constant in a different way. In particular, in each iteration it selects for further search *any* rectangle $i$ that *could* have the lowest Lipschitzian lower bound for *some* Lipschitz constant $K > 0$, subject to the constraint that the resulting lower bound be non-trivially better than the current best solution $f_{\min}$. We call the rectangles that meet these criteria "potentially optimal" rectangles, defined formally as follows:

**Definition 1** Potentially optimal hyperrectangle
Suppose we have a partition of the unit hypercube into $m$ hyperrectangles. Let $\mathbf{c}_i$ denote the center point of the $i$th hyperrectangle, and let $d_i$ denote the distance from the center point to the vertices. Let $\epsilon > 0$ be a small positive constant. A hyperrectangle $j$ is said to be *potentially optimal* if there exists some $K > 0$ such that

$$f(\mathbf{c}_j) - Kd_j \leq f(\mathbf{c}_i) - Kd_i, \quad \text{for all } i = 1, \ldots, m. \tag{3}$$

$$f(\mathbf{c}_j) - Kd_j \leq f_{\min} - \epsilon|f_{\min}|. \tag{4}$$

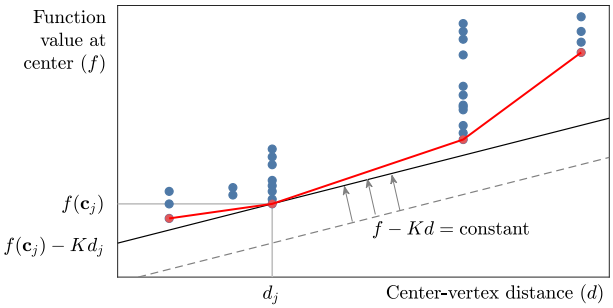


**Fig. 6** Selecting rectangles with DIRECT. Each rectangle is represented as a blue dot with horizontal coordinate equal to the rectangle's center-vertex distance and vertical coordinate equal to the function value at the rectangle's center. The rectangles meeting the first condition in the definition of potentially optimal rectangles—Eq. (3)—are highlighted in red and correspond to the lower-right convex hull of the points. (Color figure online)

A reasonable value for $\epsilon$ would be the desired relative accuracy in the solution, for example, $\epsilon = 10^{-4}$.

Figure 6 shows a diagram that helps visualize the set of rectangles meeting the first condition in the definition of "potentially optimal," that is, those rectangles satisfying Eq. (3) for some $K > 0$. In this figure, each rectangle in the partition is represented as a dot with horizontal coordinate equal to the rectangle's size ($d$) and vertical coordinate equal to the function value ($f$) at the rectangle's center. The gray dashed line in the figure shows the set of points where the rectangle lower bound, $f - Kd$, is equal to some constant. The rectangle with the *lowest* lower bound can be found by placing this dashed line below all the dots and then sliding it up until it first touches a dot. We illustrate this sliding motion in the figure, where rectangle $j$ turns out to be the one with the lowest lower bound. If we now vary the slope $K$ from zero to infinity, it is clear that the set of dots that can have the lowest lower bound—and therefore meet the first condition in the definition of potentially optimal rectangles (Eq. (3))—all lie on the lower-right convex hull of the cloud of dots, shown in the figure by the solid red line.

However, in the definition of potentially optimal rectangles, we also need to be sure that the lower bound for a rectangle is significantly better than the current best solution, that is, it must be less than or equal to $f_{\min} - \epsilon|f_{\min}|$. This condition is needed to stop DIRECT from wasting precious function evaluations selecting extremely small rectangles where we can only expect a negligible improvement. As shown in Fig. 7, the smallest selected rectangle can be found by anchoring a line on the vertical axis at $f_{\min} - \epsilon|f_{\min}|$ and rotating it up until it first touches a dot. Mathematically, this smallest selected rectangle is the one that minimizes $(f_i - f_{\min} + \epsilon|f_{\min}|)/d_i$. In the example of Fig. 7, if we apply this second condition in the definition of "potentially optimal" rectangles, then the first dot on the lower-right convex hull would be skipped. Only the rectangles associated with the three remaining dots are actually selected (shown by the solid red line).

In short, the computational procedure to select potentially optimal rectangles is to first find the smallest rectangle that can be selected as in Fig. 7 and then follow the lower-right convex hull upward. An efficient computational procedure for following the convex hull is the "Jarvis march" [28].

Putting everything together, we can summarize the DIRECT algorithm as follows:

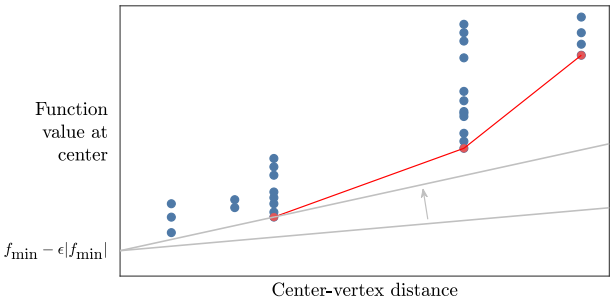


**Fig. 7** Selecting rectangles with DIRECT, continued. This figure shows how to identify rectangles meeting the second condition in the definition of "potentially optimal" rectangles—Eq. (4). This is done by rotating up a line anchored at $(0, f_{\min} - \epsilon|f_{\min}|)$ until it first touches a dot. Any rectangle smaller than this is not selected. In summary, the computational procedure for selecting potentially optimal rectangles is to first find this smallest rectangle that can be selected and then follow the lower-right convex hull upward. In this figure, the selected rectangles correspond to the red dots. (Color figure online)

**Algorithm 2** DIRECT

1: Normalize the search space to the unit hypercube. Let $\mathbf{c}_1$ be the centerpoint of this hypercube and evaluate $f(\mathbf{c}_1)$. Set $f_{\min} = f(\mathbf{c}_1)$, $m = 1$ (function evaluation counter), and $t = 0$ (iteration counter).
2: Identify the set $S$ of potentially optimal rectangles as in Fig. 7.
3: Select any rectangle $j \in S$.
4: Using the procedure described earlier (Algorithm 1), determine where to sample within rectangle $j$ and how to divide the rectangle into subrectangles. Update $f_{\min}$ and set $m = m + \Delta m$ where $\Delta m$ is the number of new points added.
5: Set $S = S - \{j\}$. If $S \neq \emptyset$ go to Step 3.
6: Set $t = t + 1$. If $t = T$, stop; the iteration limit has been reached. Otherwise go to Step 2.

DIRECT subdivides one of the largest rectangles during each iteration. Therefore, as the iterations go to infinity, the size of the largest rectangle must approach zero. As a result, the points sampled by DIRECT will be "everywhere dense"; that is, for any point $x$ in the initial hypercube and for any small $\delta > 0$, DIRECT is guaranteed to sample a point within a distance $\delta$ of $x$ after some finite (possibly very large) number of iterations. If the objective function is continuous in the neighborhood of the global optimum $x^*$, then we are also guaranteed to eventually get within any given tolerance $\epsilon$ of the global minimum function value $f^*$.

This kind of "everywhere dense" convergence is not what one would ideally want; it would be much more desirable—and more efficient—if the algorithm only converged to the global minimum instead of converging to every point. Unfortunately, if the only thing one is willing to assume about the objective function is that it is continuous, then any deterministic algorithm that guarantees convergence to the global optimum *must* sample densely.[1] To do

[1] The proof is by contradiction. Suppose there existed a deterministic algorithm $A$ that converged to the global minimum of all continuous functions without sampling densely. Now pick any one-dimensional function $f(x)$ in $[0, 1]$ and let $f^*$ be the global minimum found by $A$. Since $A$ does not sample densely, there would have to exist a point $\tilde{x}$ and a small number $\delta > 0$ with the property that $A$ never samples a point in $[\tilde{x} - \delta, \tilde{x} + \delta]$. Now define a new function $\tilde{f}(x)$ that equals $f(x)$ outside of this interval, equals $f^* - 1$ at $x = \tilde{x}$ and, for any point $x'$ in $(\tilde{x} - \delta, \tilde{x} + \delta)$, interpolates linearly between the assumed value of $f^* - 1$ at $\tilde{x}$ and the value of $f(x)$ at the nearest interval end point. If $A$ were applied to this new function $\tilde{f}(x)$, it would behave exactly as it did with $f(x)$ because $A$ is deterministic and never samples inside the interval $[\tilde{x} - \delta, \tilde{x} + \delta]$. However, this new function $\tilde{f}(x)$ is continuous and has a minimum equal to $f^* - 1 < f^*$, so $A$ would not find the global minimum of $\tilde{f}(x)$. The contradiction proves that any deterministic algorithm that converges to the global minimum of all continuous functions must sample densely.

better than "everywhere dense" convergence, one must assume more about the objective function than mere continuity. For example, one could assume knowledge of an upper bound on the magnitude of the first or second derivatives, or perhaps knowledge of the functional form that would enable computing lower bounds over subregions (e.g., with interval analysis). However, since we are assuming that the function is a continuous black-box, DIRECT's "everywhere dense" convergence is as good as it gets.

### 2.2 Strengths of the DIRECT algorithm

The DIRECT algorithm has several strengths. First, it is a black-box algorithm, so there is no need to assume the availability of gradients or other special information about the objective function. Second, as just mentioned, it is guaranteed to converge to the global minimum if the function is continuous [30]. Third, the algorithm is deterministic, so there is no need for multiple runs. Fourth, in each iteration, DIRECT usually selects several rectangles for further search, and all the associated function evaluations can be done in parallel to speed up the search. Fifth, and most importantly, DIRECT has no hyperparameters that set the balance between local and global search. As discussed above, this is accomplished by selecting all those rectangles that would have the lowest lower bound for *some* Lipschitz constant, where small constants select rectangles good for local search, and large constants select those good for global search. In contrast, other algorithms have explicit parameters that set this balance (e.g., the population size, selection bias, and mutation rate for genetic algorithms), and additional effort is required to tune the parameters for the problem at hand. By avoiding such parameters, DIRECT is able to be more robust, in the sense that it can be expected to work well across problems of varying difficulty.

While DIRECT does have *one* hyperparameter, the desired accuracy $\epsilon$, this parameter's purpose is not so much to set the overall local-global balance as it is to make sure that DIRECT does not get *too* local in its search, wasting function evaluations in pursuit of very small improvements. Given that $\epsilon$ can be interpeted as the desired accuracy, the parameter has intuitive meaning and can be easily specified. However, as we will see later, DIRECT's use of the $\epsilon$ parameter can have negative effects on performance and leaves room for improvement.

Another strength of DIRECT is that the search points tend to cluster around the global minima—a property that can be used to identify good places from which to launch a local optimizer to fine-tune the solution. Two good examples are shown in Fig. 8. On the left, we show the contours of the Branin test function, which has three global minima; after 247 evaluations, the points sampled by DIRECT clearly cluster around these locations. On the right, we show the contours of the six-hump camel test function, which has two global minima and, once again, the points sampled by DIRECT cluster around the two global minima.

In a recent application of DIRECT to genetics (looking for chromosome positions that correlate with observed traits), Ljungberg et al. [42] compared DIRECT to a genetic algorithm in terms of how well the sampled points clustered around the four largest peaks in the objective function. The results after 6000 evaluations for the two-variable version of their problem are shown in Fig. 9. The left side shows the points sampled by DIRECT, which clearly cluster around the four largest peaks (numbered 1–4 in the figure); the right shows the points sampled by the genetic algorithm, which are not as sharply clustered. Of course, the performance of the genetic algorithm strongly depends on the specific implementation (e.g., genetic algorithms using the "niching" technique are designed to identify multiple local optima and might perform better). As we discuss later, this clustering property has been exploited in one of the more successful extensions of DIRECT [26].

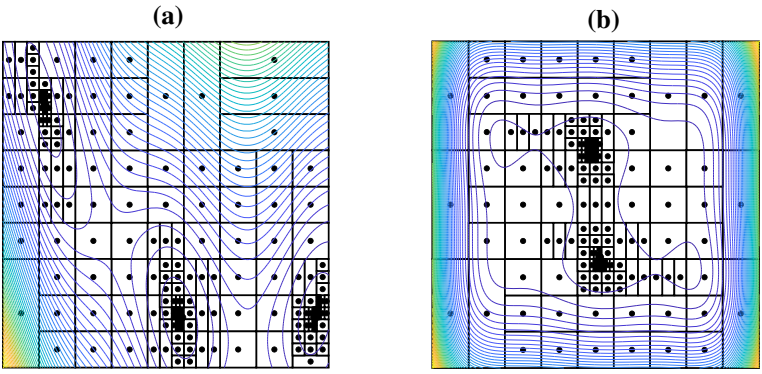


**Fig. 8** Sampled points for DIRECT cluster around the global minima for the Branin test problem (left) and the six-hump camel test problem (right)

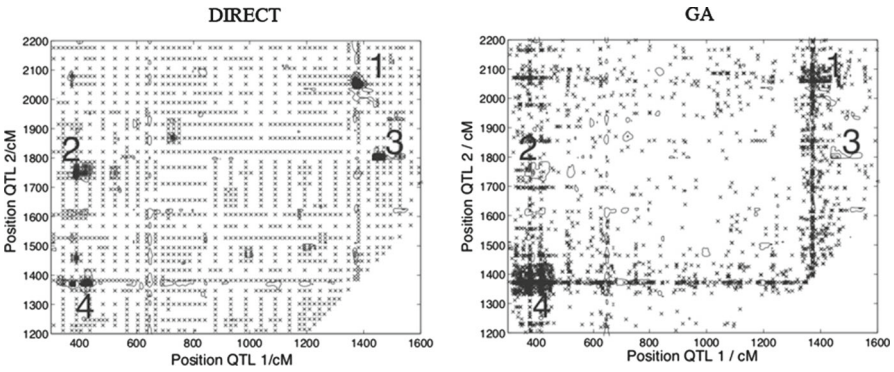


**Fig. 9** Comparison of points sampled by DIRECT and a genetic algorithm in a genetics study. The points from DIRECT cluster more clearly around the four peaks of the objective function. Reproduced from Ljungberg et al. [42]

## 2.3 Weaknesses of the DIRECT algorithm

### 2.3.1 Low convergence rate in solution refinement

Recall that DIRECT usually selects several rectangles per iteration, some good for local search, some for global search, as well as others in between. This has the advantage that, as soon as the global part of the search finds the basin of the global minimum, the local part of the search immediately fine tunes it. The problem is that, while this local refinement is going on, DIRECT is still doing a lot of global search, and this global search puts a "drag" on the speed of the local refinement. As a result, DIRECT can be fast to find the basin of the global optimum, but be slow to refine the solution to high accuracy.

To illustrate this weakness, consider the problem of minimizing the function $f(x_1, x_2) = 1 + x_1 + x_2$ on the unit hypercube. Clearly, the global minimum occurs at the point $(0, 0)$ where $f = 1$. DIRECT (using $\epsilon = 10^{-4}$) gets to within 1% accuracy of this solution in 90

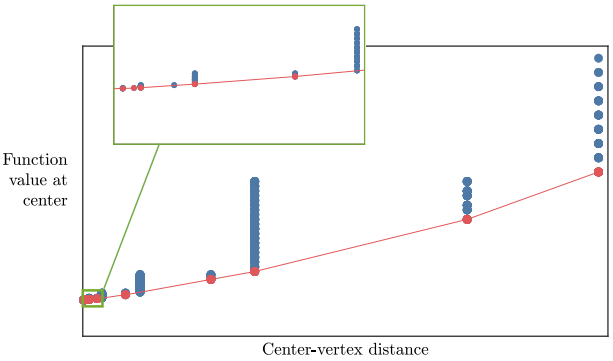


**Fig. 10** Centerpoint function value versus center-vertex distance for the linear function $1 + x_1 + x_2$ after 497 function evaluations

function evaluations.[2] But it takes fully 616 evaluations to get to a solution within 0.01% accuracy (i.e., $f \leq 1.0001$). Why does this happen? Well, there are two reasons.

The first reason is illustrated in Fig. 10 which shows our usual selection diagram after 16 iterations and 497 function evaluations. As we can see, the dots for the rectangles group themselves into vertical columns or "levels" corresponding to rectangles of the same size (same center-vertex distance). In each column, only the lowest dot (corresponding to the rectangle with the best function value) can possibly be selected—and even then, it is only selected if it is on the lower-right convex hull. In this case, there are 10 levels with selected rectangles, shown as filled red dots in the figure (some of the markers for very small center-vertex distances overlap in the main figure, but are expanded in the inset). This means that, in addition to sampling the rectangle with the global optimum, nine others are sampled for global search, creating a drag on the refinement of the solution. As the search continues and more rectangles are divided, the *range* of rectangle sizes—the number of levels in Fig. 10—increases, so this global drag will potentially become even stronger as the search proceeds.

In this example, the global drag is especially high because, for each level with a selected rectangle (filled red dot in Fig. 10), there are actually several rectangles that are *tied* for the best function value and hence selected. These ties are illustrated in Fig. 11 which shows the rectangles in the partition, highlighting the selected rectangles in blue. As we can see, there are sets of selected rectangles of the same size that line up on diagonals. Within each set, all the rectangles have the same value of the objective $1 + x_1 + x_2$ and hence are tied. The net result is that not just 10 rectangles are selected, corresponding to the 10 red dots in Fig. 10, but 38 rectangles are selected when all ties are considered (not all are visible in the figure, because the small ones in the lower left corner are hard to see). In this case, we do select and subdivide the rectangle with the global minimum, but we also select and sample 37 others, thereby creating a huge "global drag" on local refinement. This is why DIRECT requires 616 function evaluations to refine the solution to 0.01% accuracy.

Of course, a linear function such as $f(x_1, x_2) = 1+x_1+x_2$ is trivial to optimize and would not be the kind of problem for which one would select DIRECT as the optimizer. We have

[2] All the results reported here use Eq. (5) in the definition of "potentially optimal" rectangle. We report the number of evaluations when the accuracy is first achieved assuming that, within each iteration, the potentially optimal rectangles are sampled from the smallest to the largest. This approach differs from the original
Footnote 2 continued
DIRECT article [30] which always reported the evaluation count *at the end of the iteration* in which the accuracy was first achieved.

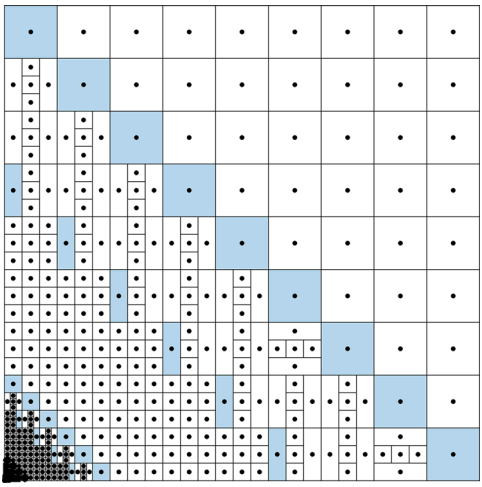

**Fig. 11** Selected rectangles (shaded) after 497 function evaluations when minimizing $1 + x_1 + x_2$

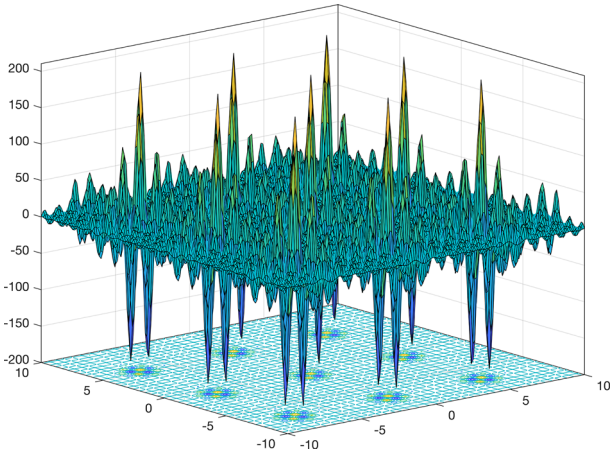


**Fig. 12** Shubert test function

used it here to drive home the point that one does not need a complicated and exceptionally unusual problem to cause DIRECT to perform poorly due to the weakness of global drag. As we will see, some of the later modifications to DIRECT specifically address this issue of global drag on local refinement.

### 2.3.2 Search around local minima delays finding global minimum

When DIRECT converges slowly, a common reason is that it stumbles upon a suboptimal local minimum early in the search, before the basin of the global minimum is found. In such cases, DIRECT can sometimes spend an excessive number of function evaluations refining this suboptimal local solution to high accuracy, slowing down the discovery of the true global minimum. An extreme case of this problem happens when solving the two-variable Shubert test function using $\epsilon = 0$. The Shubert function is a strange test function with many peaks and valleys, 18 of which are global minima (see Fig. 12).

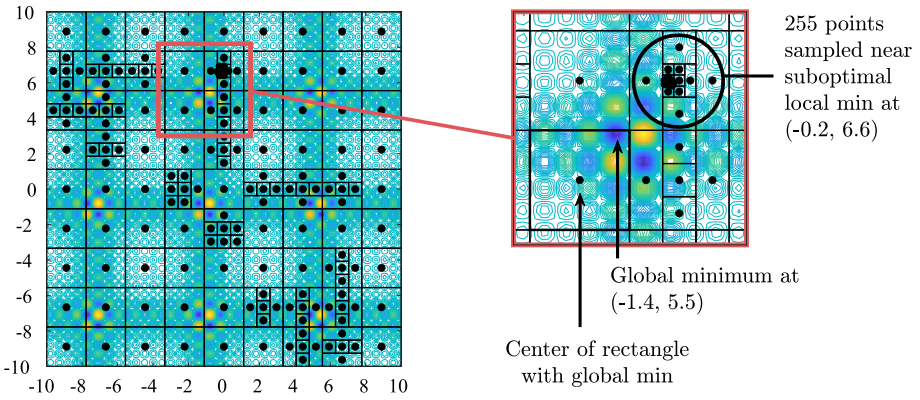


**Fig. 13** DIRECT with $\epsilon = 0$ after 415 function evaluations on the Shubert test function

When DIRECT is applied to the Shubert problem using $\epsilon = 0$, it quickly finds a suboptimal local minimum at $(-0.2, 6.6)$. As the search proceeds, the sampled points tend to cluster around this first optimum; Fig. 13 shows the sampled points after 25 iterations and 415 function evaluations. The problem is that this first local minimum is not a *global* minimum and DIRECT is over-refining the solution, creating super-small rectangles with center-vertex distances as small as $10^{-7}$. In fact, of the 415 evaluations shown in Fig. 13, DIRECT has sampled 255 points in the neighborhood of the suboptimal local and only 160 elsewhere. As we can see in figure, one of the global minima is nearby, but unfortunately it lies in a bigger rectangle whose centerpoint function value happens to be poor. As a result, it may take many iterations before the rectangle containing the global minimum is selected and subdivided.

In Fig. 14 we take a closer look at what is happening with the selection of rectangles after 415 evaluations. All together, 13 rectangles are selected, but 11 of them have center-vertex distances that are so small – all less than 0.0002 – that they overlap in the plot of center-point function value versus rectangle size. All these small rectangles are clustered around the local minimum. Only two big rectangles are selected (in the figure we can only see one red dot with a big center-vertex distances, but actually this dot represents two tied rectangles, that is, two rectangles with the same size and center function value).

In Fig. 14, we have also used a green dot to identify the rectangle with the global minimum at $(-1.4, 5.5)$. As shown in the figure, the center of this rectangle happens to have a poor function value; as a result, the dot with the global minimum is not near the bottom of its column, and so it may take quite a few iterations (in which the rectangles below it in the column are processed) before this rectangle is selected and explored further.

In this case, we can also see that DIRECT's global search can be insufficiently thorough. For example, it might also make sense to select the rectangles labeled in the figure as "not selected but nondominated." These rectangles are not strictly on the lower-right convex hull of the dots, and so are not selected, but they represent efficient tradeoffs between rectangle size and function value in the sense that they are Pareto optimal.

Summing up, there are two reasons why DIRECT can have a long delay in finding the global min: first, it may waste function evaluations over-refining a suboptimal local minimum; second, the global search that it does do in each iteration may be insufficient. As we will see later, some of the suggested modifications to DIRECT address each of these reasons for the potential delay in finding the global minimum.

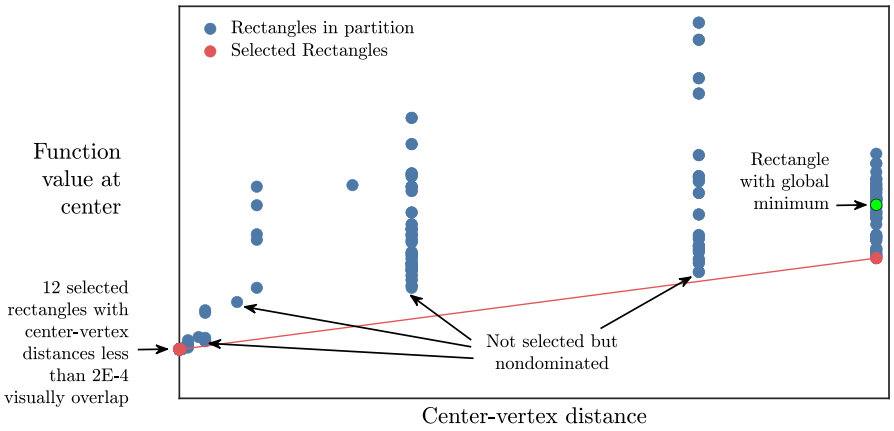


**Fig. 14** The status of rectangle selection on the Shubert test function after 415 function evaluations when using DIRECT with $\epsilon = 0$

The problem of over-refining a suboptimal local minimum was partially addressed in the original DIRECT algorithm by introducing the parameter $\epsilon > 0$ in the definition of potentially optimal rectangles—see Eq. (4). In Fig. 7, we already saw how this can prevent some of the smaller rectangles on the lower-right convex hull from being selected. Intuitively, when we set (say) $\epsilon = 10^{-4}$, we discourage DIRECT from exploring regions of the search space where it seems unlikely to improve upon the solution by at least a fraction $\epsilon = 10^{-4}$.

For example, on the two-variable Shubert test function DIRECT, if we set $\epsilon = 0$ then DIRECT takes over 10,000 evaluations to refine the solution to 0.01% accuracy. If we set $\epsilon = 10^{-7}$, the number of evaluations shrinks to 5713. And if set $\epsilon = 10^{-4}$, the required evaluations shrink further to 2933. Thus, using a small, positive $\epsilon$ can help accelerate the refinement of the solution.

Unfortunately, introducing the $\epsilon$ parameter, while preventing excessive refinement of a suboptimal local minima, also has a negative side effect: once the basin of global minimum has been found, the use of $\epsilon > 0$ slows the refinement of the global minimum to very high accuracy. So while the use of $\epsilon > 0$ does help avoid excessive local search, it only partly addresses the problem, and some of the enhancements we discuss later introduce other mechanisms to address this weakness of DIRECT.

### 2.3.3 Failure to exploit local trends

In DIRECT, each rectangle is evaluated for further search based only on its centerpoint function value and its size; any trends that exist *between* rectangles are ignored. Thus, in the case of the function $1 + x_1 + x_2$ discussed above, DIRECT does not detect or exploit the linearity or monotonicity of the function.

Another example is the Branin function shown on the left in Fig. 9. This is a two-variable problem, and DIRECT finds the solution to within 1% accuracy in only 51 evaluations. But if we add a third variable $x_3$ that has no effect on the function value, then DIRECT takes 839 evaluations to find the solution to within 1% accuracy. Even though the third variable has no effect, DIRECT isn't able to sense this fact and explores the third dimension with equal vigor.

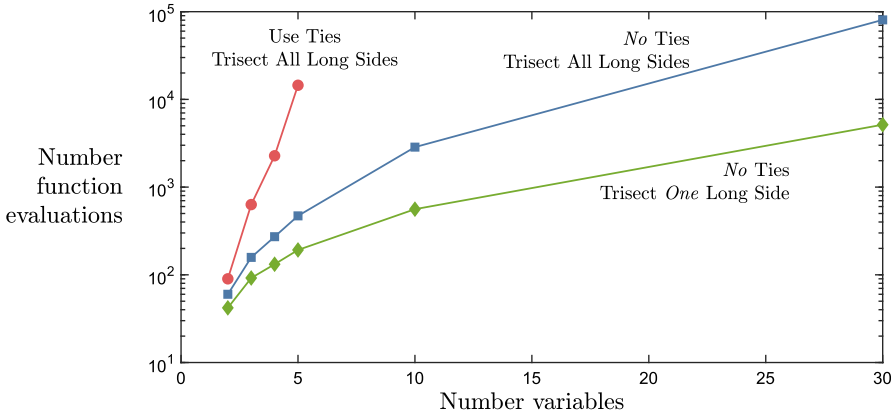


**Fig. 15** Function evaluations to be within 1% accuracy when minimizing $1 + x_1 + x_2 + \cdots + x_n$ plotted against the number of variables $n$ for three ways to do rectangle selection and subdivision

### 2.3.4 Performance degradation in higher dimensions

Like many other optimization algorithms, the DIRECT method is not immune to the "curse of dimensionality." As the number of dimensions increase, the number of function evaluations required by DIRECT to converge to a solution tends to increase dramatically. Intuitively, in $n$ dimensions, in order to isolate the global optimum in a small box of side length $3^{-8} = 0.00015$, we would have to trisect the box containing the global minimum at least 8 times on each dimension for a total of $8n$ times. The problem is that, in each iteration, we do not just select and trisect the rectangle containing the global optimum; instead, we also select other rectangles for global search—and there can be *many* others, especially if, as in the linear example, there are ties for the best rectangle at each level.

The impact of problem dimension on convergence speed can be seen dramatically in Fig. 15 which shows the number of function evaluations to converge within 1% accuracy for the function $1 + x_1 + x_2 + \cdots + x_n$ plotted versus the number of variables $n$. Three different curves are shown, each using a different method for rectangle selection and subdivision. The red curve shows the original algorithm that samples along all long sides and accepts ties for potentially optimal rectangles. With this method of selection and sampling, the problem with just five variables already takes 14,492 function evaluations. This is clearly excessive for such a simple function.

In his revision of DIRECT, Jones [29] suggested two modifications to reduce the global drag. The first modification is the obvious one: just select one rectangle from each level and ignore any ties for being "potentially optimal." The impact of this change is shown by the blue curve, where the function evaluations to solve the five variable problem drop from 14,492 to just 470. The second modification was to trisect a selected rectangle on only *one* long side, picking the long side corresponding to a variable that has been split the least over the entire search.[3] By trisecting only on one long side, we increase the number of levels

[3] The goal here is to break ties for the longest side in a way that doesn't favor one dimension over another, so as not to be "arbitrary." Of course, one could break the tie randomly, but that would cause DIRECT to lose the desirable property of being deterministic. What we do, instead, is keep track of how many times we trisect *any* rectangle on each dimension and then, when presented with a tie for longest side, we choose the dimension that has been trisected the least during the search to this point (if there is still a tie, we then favor the lower indexed variable).

(distinct rectangle sizes) by at most one; in contrast, when we subdivide on all long sides as in the original algorithm, we can increase the number of levels by up to $n$ and, the more levels there are, the greater the global drag can be. The green curve shows what happens if we do not pick ties *and* we also trisect the rectangle on only one long side. This further reduces the function evaluations to solve the five-variable problem to 192 evaluations.

#### 2.3.5 Inflexibility of the partitioning scheme

DIRECT itself, without the use of local optimizers or other enhancements, can only sample points that are centers of hyperrectangles in the partition. As a result, some points cannot be sampled. An often-mentioned example are points on the boundary of the feasible space, which can only be approached arbitrarily closely, but never sampled exactly. In fact, several authors have mentioned that DIRECT can be especially slow to converge to an optimum on the boundary [27,36].

This inflexibility also makes it difficult for DIRECT to accept a user-defined starting point, even if a good one was known (though this can be done via problem reformulations, as we discuss in Sect. 3.10). In any case, it is certainly not possible to accept a set of *several* initial points, such as a Latin hypercube. Similarly, even if some analysis suggested that a certain point was likely to be the optimum (e.g., the origin in Fig. 11), there is no easy way to add that point to the search and still maintain the property that every sampled point is the center of a rectangle in the partition. As we will see, other partitioning schemes, such as that used by Liu et al. [36], provide more flexibility.

## 3 Modifications to the DIRECT algorithm

In this section we survey a number of articles that have modified or extended DIRECT. For each article, we identify the weaknesses that are addressed.

### 3.1 Making DIRECT insensitive to additive and multiplicative scaling

Recall that, for rectangle $j$ to be "potentially optimal" and selected for further search, we required two things. First, there had to exist a Lipschitz constant $K > 0$ such that the lower bound for the rectangle was the same or lower than that of any other rectangle:

$$f(\mathbf{c}_j) - Kd_j \leq f(\mathbf{c}_i) - Kd_i, \quad \text{for all } i = 1, \ldots, m. \tag{3}$$

Second, the lower bound had to be non-trivially better than our current best solution $f_{\min}$. We made this second requirement precise by requiring

$$f(\mathbf{c}_j) - Kd_j \leq f_{\min} - \epsilon |f_{\min}|. \tag{4}$$

One weakness of this approach, discovered by Sergeyev and Kvasov [63], occurs when there is a suboptimal local minimum with a function value close to zero. In this case, if DIRECT gets close to this local minimum, the term $\epsilon |f_{\min}|$ will be very small, which again allows excessive local search.

Another issue with Eq. (4), pointed out by Finkel and Kelley [14], was that this equation makes DIRECT sensitive to additive scaling of the objective function; that is, adding a constant $a$ to the objective function can change the points sampled by DIRECT, even though adding a constant to the objective function does not change the problem in any fundamental

way. To see why this is so, suppose one adds a very big number $a$ to the objective function. If $a$ is large enough, then the value of $|f_{\min}|$ in Eq. (4) would increase and, hence, we would be requiring the rectangle lower bound to be below the current best solution $f_{\min}$ by a greater amount. As a result, the set of selected rectangles might change (we might skip selecting some of the smaller rectangles). To make DIRECT insensitive to additive scaling, Finkel and Kelly suggest replacing Eq. (4) with

$$f(\mathbf{c}_j) - Kd_j \leq f_{\min} - \epsilon(f_{\text{median}} - f_{\min}), \tag{5}$$

where $f_{\text{median}}$ is the median of all the function values sampled so far during the search. With this change, adding $a$ to the objective function has no effect, as both $f_{\text{median}}$ and $f_{\min}$ would increase by $a$ and so the difference $f_{\text{median}} - f_{\min}$ would remain unchanged. Note that, while the original version of DIRECT was sensitive to *additive* scaling, it was insensitive to *multiplicative* scaling. That is, it produces the same set of iterates if we multiply the objective function by some positive constant $b$. To understand why this is the case, suppose rectangle $j$ were potentially optimal, that is, assume it satisfies Eqs. (3) and (4) for some $K > 0$. If we then multiply the objective function by $b > 0$, rectangle $j$ would still satisfy Eqs. (3) and (4) if we use the Lipschitz constant $bK$, and hence it would still be potentially optimal. In summary, with the change suggested by Finkel and Kelly, DIRECT produces the same sequence of iterates when applied to $f(x)$ as it does when applied to $a + bf(x)$ for any $a$ and any $b > 0$. This is an obviously desirable property called "strong homogeneity" [12,66,78].

### 3.2 Locally-biased DIRECT (DIRECT-l)

Gablonsky [17] and Gablonsky and Kelly [18] suggested two changes to rectangle selection to reduce global drag on refining local solutions. The first was one we have already discussed: selecting only one of several rectangles tied for being "potentially optimal." The second was to redefine a rectangle's size to be half the length of the longest side, as opposed to the Euclidean distance between the center point and the vertices. This corresponds to using the infinity norm in the definition of the Lipschitz constant in Eq. (2). If we use half the longest side as a measure of the rectangle size, two rectangles with *different* center-vertex distances—and hence different size in the original DIRECT algorithm—can have the *same* size if their longest sides have the same length. An example is shown in Fig. 16, where the two shaded rectangles have different center-vertex distances but have the same longest side, and so would be considered to be the same size with Gablonsky's approach. Overall, in Fig. 16, there are rectangles with five different center-vertex distances, but there are only three different long-side lengths.

With Gablonsky's modification, in our usual selection diagram, there will be fewer levels (distinct rectangle sizes) and hence less "global drag" on local refinement. This is illustrated in Fig. 17, which compares the selection diagrams for the linear function $1 + x_1 + x_2$ after 500 function evaluations for the original DIRECT algorithm (left) and for DIRECT with Gablonsky's modification (right). As expected, there are fewer distinct rectangle sizes when using Gablonsky's modification, and so there is less global drag on local refinement. Gablonsky calls the method "locally biased DIRECT" or DIRECT-l.

The reduced global drag translates to faster convergence to the global minimum and a reduced "curse of dimensionality" on problems (such as the linear example), in which the basin of the global optimum is found fairly easily. This is illustrated in Fig. 18, which shows how the addition of Gablonsky's modification further speeds up convergence on the

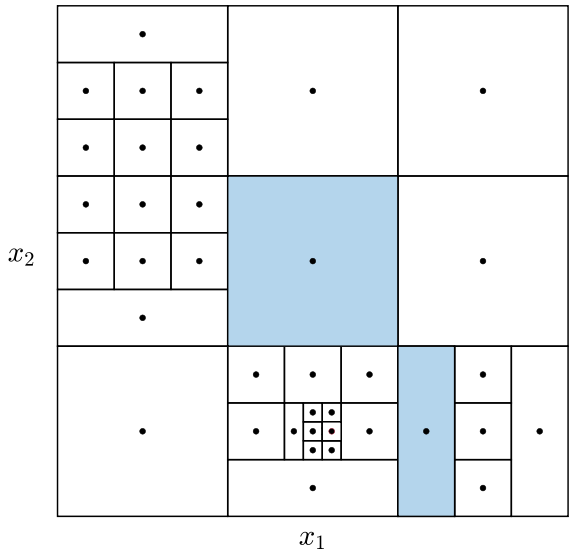


**Fig. 16** The two shaded rectangles have the same size if we use the approach by Gablonsky and Kelly [18], in which the size of a rectangle is equal to half its longest side

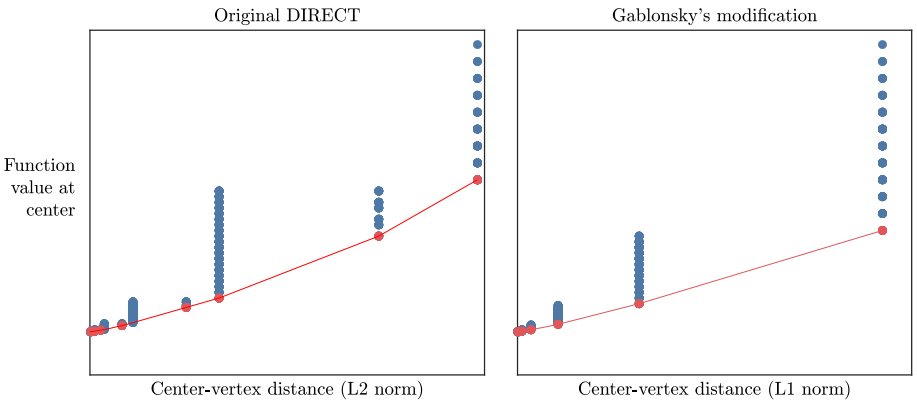


**Fig. 17** Selection diagrams for the linear function $1 + x_1 + x_2$ after 500 function evaluations for the original DIRECT algorithm (left) and the version with Gablonsky's modification (right)

linear function beyond what we already achieved by ignoring ties for the potentially optimal rectangles and only trisecting on one long side.

In summary, Gablonsky introduced two modifications to the original DIRECT algorithm: first, he ignores ties for potentially optimal rectangles, just picking one arbitrarily. Second, he redefines the size of a rectangle using the infinity norm instead of the Euclidean two-norm. On the test problems used in the original DIRECT paper (2 to 6 dimensions), he finds that his DIRECT-l modification generally outperforms the original DIRECT, though in some cases only marginally. But as we can see from Fig. 18, the approach can greatly reduce the curse of dimensionality on higher-dimensional problems.

There is, however, a drawback to Gablonksy's modification for how to measure rectangle size. This modification creates less "global drag" by doing less global search, and so one can expect worse performance on very difficult problems where global search is critical. Sergeyev and Kvasov [63] discovered several such examples when they compared DIRECT

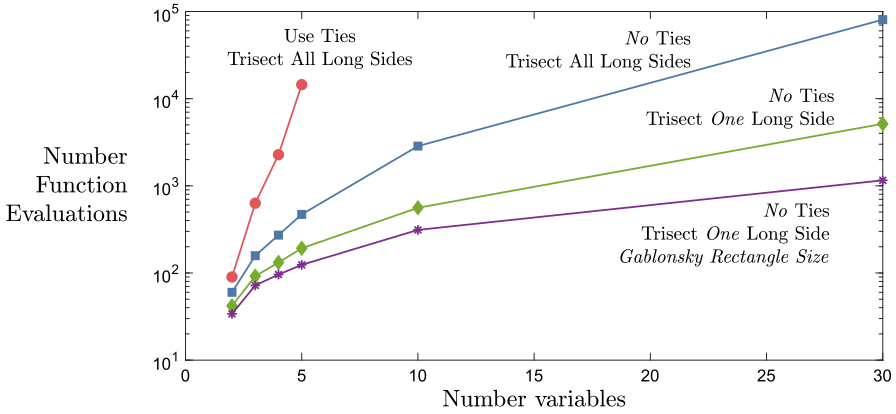


**Fig. 18** Function evaluations to be within 1% accuracy when minimizing $1+x_1+x_2+\cdots+x_n$ plotted against the number of variables $n$ for four ways to do rectangle selection and subdivision. Best results are achieved by ignoring ties for potentially optimal rectangles, only trisecting on one side, and using Gablonsky's modified definition of rectangle size

and DIRECT-l on their "GLKS" test suite of low-dimensional ($2 \leq n \leq 5$), extremely hard, multimodal test problems [19,63]. However, on problem classes of known low difficulty, using Gablonsky's rectangle-size modification seems to be a reasonable hedge against the curse of dimensionality.

### 3.3 Adaptive setting of the $\epsilon$ parameter (DIRECT-restart)

In Sect. 2.3.2, we showed that setting $\epsilon = 0$ could lead to excessive local search that delayed the discovery of the global minimum. In the original DIRECT algorithm, we addressed this problem by setting $\epsilon$ to a small positive value (e.g., $10^{-4}$), which discourages the selection of very small rectangles and makes the search more global. However, as we mentioned before, this was an incomplete solution, because using a positive value of $\epsilon$ also slows down the refinement of the global solution once it has been discovered.

Finkel and Kelly [15] attempt to resolve this issue by making the setting of $\epsilon$ *adaptive*. They start with $\epsilon = 0$ and begin the iterations. As long as the search is finding improved solutions, they maintain $\epsilon = 0$. However, if five consecutive iterations pass with no improvement in the best function value, they conclude that the search may be stagnated around a local optimum. To prevent excessive local search, they then set $\epsilon = 0.01$, which makes the search more global. They maintain $\epsilon = 0.01$ until the search finds an improved solution, in which case they again set $\epsilon = 0$ to promote the local refinement of this new local minimum. They also stop using $\epsilon = 0.01$ (and switch to $\epsilon = 0$) if 50 iterations pass without improvement, as this may indicate that the global minimum has been found and one should work on refining it to higher accuracy. Finkel and Kelly call this method DIRECT-restart because it "restarts" the iterations with different values of epsilon.

We have implemented DIRECT-restart where we define "improvement" of the solution to mean that we find a rectangle with function value $f$ for which we have

$$f_{\min} - f \geq 10^{-4} |f_{\text{median}} - f_{\min}|. \tag{6}$$

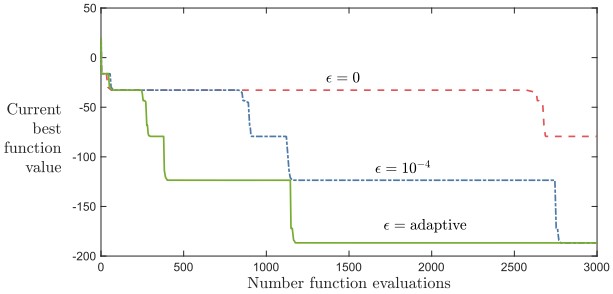


**Fig. 19** Iteration history of three versions of DIRECT on the Shubert test function: DIRECT with $\epsilon = 0$, DIRECT with $\epsilon = 10^{-4}$, and DIRECT with the adaptive control of $\epsilon$ as in Finkel and Kelly [15]

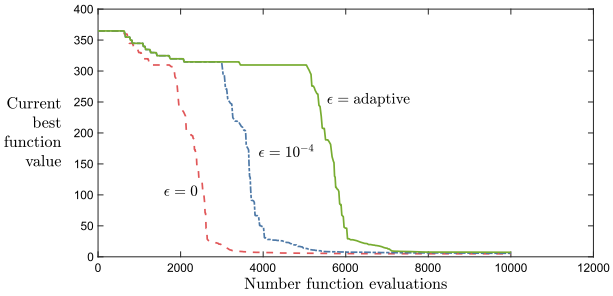


**Fig. 20** Iteration history of three versions of DIRECT on the Powell test function with 48 variables on $[-4, 5]^{48}$: DIRECT with $\epsilon = 0$, DIRECT with $\epsilon = 10^{-4}$, and DIRECT with the adaptive control of $\epsilon$ proposed by Finkel and Kelly [15]. In this case, the adaptive control of epsilon yields poorer convergence

This equation states that the improvement over the previous best solution must be at least $10^{-4}$ times the spread between the current median and min function values. Figure 19 compares the performance of three variations of DIRECT on the Shubert test function. All three versions trisect rectangles on only one long side, ignore ties for potentially optimal rectangles, and use Eq. (5) in the definition of potentially optimal rectangles. The versions differ only in how the parameter $\epsilon$ is set: the first uses $\epsilon = 0$, the second uses $\epsilon = 10^{-4}$, and the third uses the adaptive approach just discussed.

As we can see in Fig. 19, with $\epsilon = 0$ DIRECT spends so many evaluations refining local minima that it ends up far from the optimum after 3000 evaluations. Setting $\epsilon = 10^{-4}$ allows DIRECT to find the global minimum within 3000 evaluations. DIRECT-restart also finds the global minimum, but it does so with fewer function evaluations because it wastes less time exploring suboptimal local minima.

Given this encouraging result, we decided to try DIRECT-restart on seven test functions identified by Liu et al. [39], which were stated to have the property that setting $\epsilon = 0$ led to poor convergence. We found that DIRECT-restart performed better than DIRECT with $\epsilon = 0$ on one problem (Shubert), performed about the same on five problems, but performed worse on the Powell test function in 48 dimensions on $[-4, 5]^{48}$. The results for the Powell problem are shown in Fig. 20. For this problem, the results were exactly the opposite: DIRECT-restart performed *worse* than the other two variants.

The poor performance of DIRECT-restart on the Powell problem is due to the adaptive method's going back to global search (due to five iterations with no improvement) when it would have been better to continue local refinement using $\epsilon = 0$. This result shows

just how difficult it is to develop a method that works well across problems with different characteristics. In this case, the different characteristic was how many iterations of local search are typically needed to improve the objective. Because the Powell problem has higher dimensionality than the Shubert problem (48 variables vs. 2), it is understandable that local search (i.e., using $\epsilon = 0$) might need more iterations to improve the objective. This insight suggests a simple modification: make the number of "no improvement iterations" that triggers a switch to global search increase with the number of variables according to some formula. This may be worth a try, but there is almost certainly no formula that will work well with all problems. The goal here, as with all modifications to DIRECT, is to improve performance across a wide range of problems of the sort one expects to encounter in real-world applications.

### 3.4 Pareto-Lipschitzian optimization (PLO and PLOR)

As mentioned previously, one of the weaknesses of DIRECT is that the global search can sometimes be insufficiently thorough, thereby delaying the discovery of the global minimum. We saw this in Fig. 14, where some rectangles were not dominated on function value and size, but nevertheless were not selected because they were not on the *lower-right convex hull* of points in our selection diagram. The Pareto-Lipschitzian optimization algorithm (PLO) introduced by Mockus [45] addresses this weakness in a straightforward manner: it simply selects all the rectangles that are non-dominated (Pareto optimal) with respect to center-point function value and rectangle size. Fig. 21a, b contrast the usual DIRECT selection method with the Pareto method. In Fig. 21a, we show the rectangles that would be selected by DIRECT (those on the lower-right convex hull); in this case, three rectangles are selected. In Fig. 21b, we show the rectangles selected by the Pareto method; in this case five rectangles are selected. Because the points on the convex hull are always nondominated, the points selected by DIRECT are always a subset of those selected by the Pareto method. Hence, in general, the Pareto method selects more rectangles and does a more global search.

The Pareto approach has several attractive features. First, it takes better advantage of parallel computing because it selects more rectangles than DIRECT. Second, the Pareto approach only depends on *relative* function values; as a result, one would get the same search if one used any transformation of the objective function that preserves the relative ordering of the function values (e.g., log, square root, exponential). This is desirable because sometimes the scale of the objective is arbitrary (e.g., noise could be measured in sound pressure or decibels). Finally, the Pareto approach does not have any hyperparameters, such as the $\epsilon$ in DIRECT.

However, the Pareto approach is not without weaknesses. First, by eliminating the $\epsilon$ parameter, the method loses any mechanism to prevent the selection of very small rectangles and this, in turn, can lead to excessive local search around suboptimal local minima. Mockus et al. [46] mention that excessive local search is occasionally a problem for the PLO algorithm. A second potential weakness of the PLO selection method is that it has more "global drag" on fine-tuning local minima because it does more global search than DIRECT. Therefore, we can expect it to be slow when refining solutions to high accuracy.

Mockus et al. [46] propose to reduce this global drag by only selecting the first and last rectangle on the Pareto front, as shown in Fig. 21c. They call this variation "reduced-set Pareto-Lipschitzian optimization" or PLOR. The first rectangle is the one with the best function value at the center (breaking any ties in favor of larger rectangles), and the second rectangle is the largest rectangle with the best function value. This clearly reduces global search and we can expect the search to converge faster when more global search is

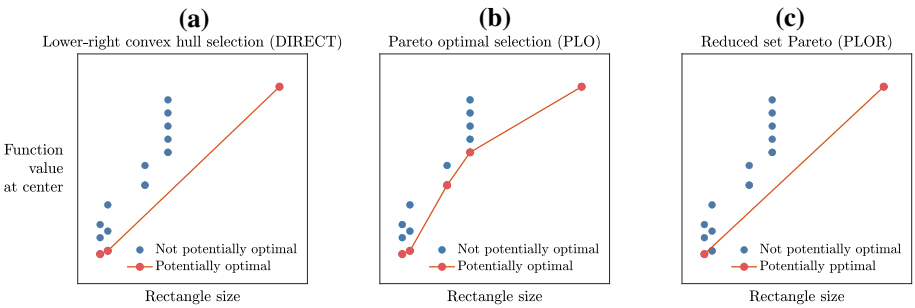


**Fig. 21** Comparison of selection methods: DIRECT, Pareto-Lipschitzian (PLO), and reduced-set Pareto-Lipschitzian (PLOR)

not required. In numerical work on the low dimensional test problems used in the original DIRECT paper (2 to 6 dimensions) [30] and on some small truss structural optimization problems, the authors do indeed find that PLOR often converges faster than DIRECT.

However, the reduced amount of global search can occasionally cause PLOR to require an excessive number of evaluations to find the basin of the global minimum. Mockus et al. [46] only found this to happen for one of their test problems (Shekel 5), but when we replicated the method in our own code, we saw it happen for at least two others (Branin and Six-Hump Camel). Conceptually, insufficient global search is a clear risk with PLOR: If the global minimum is in a medium-sized rectangle (not smallest, not largest), then the reduced amount of global exploration in PLOR may delay finding the global minimum for many iterations.

### 3.5 Revision of DIRECT

Jones [29] published a revision of DIRECT with enhancements to reduce the global drag, two of which we already mentioned in Sect. 2.1: (1) trisect a rectangle on only *one* long side and, (2) only select one of several tied potentially optimal rectangles. As illustrated in Fig. 18, this further reduced global drag on our simple linear test function.

In addition to these two enhancements, Jones [29] introduced a modification to help DIRECT speed up the refinement of solutions to high accuracy. The idea was simple: to combine DIRECT with a good local search algorithm. We start by running DIRECT for some predetermined number of function evaluations (say 100) and then fine tune the current best solution with the local optimizer. Having updated the value of $f_{\min}$ based on the local search, we then return to the regular DIRECT iterations, looking for a solution that is even better than the local minimum just found. However, DIRECT does not proceed the same as it would have without the local optimizer. Instead, the search is more global because the local optimizer reduces the value of $f_{\min}$, which makes the search more global by ignoring very small rectangles (see Fig. 7). DIRECT will now be looking for a point that improves upon the local solution—in effect, it will be looking for the basin of convergence of a better local minimum. If DIRECT finds such an improving point, then we run a local search from this point and again return to DIRECT. This process continues, alternating between DIRECT and the local optimizer, until we reach a pre-determined limit on the total number of function evaluations (for both DIRECT and the local optimizer). Used in this way, DIRECT becomes an intelligent routine for selecting starting points for the local optimizer.

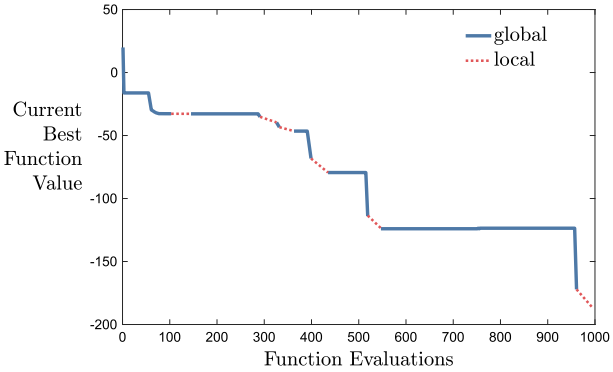


**Fig. 22** Convergence of the DIRECT with local optimizer on the Shubert test function

There is one other benefit of this approach: When using DIRECT with a local optimizer, we can also afford to use a higher value for $\epsilon$, say $10^{-2}$. The larger value of $\epsilon$ further prevents excessive local search around a suboptimal local minimum, and we can do this because we rely on the local optimizer (not DIRECT) to refine solutions to high accuracy.

Figure 22 shows how this hybrid global-local approach works on the Shubert test function using $\epsilon = 10^{-2}$. The local optimizer used was the derivative-free algorithm BOBYQA [56] using the available Python implementation.[4] We alternate between global and local search six times, getting to the global minimum within 0.01% accuracy in 995 function evaluations, less than the 2967 required by the original DIRECT algorithm.

As we will see later in Sects. 3.10, 3.11, and 3.12, the simple and almost obvious idea of hybridizing DIRECT with a local optimizer has appeared in other DIRECT variants. One of the key differences distinguishing these variants is the criterion for *when* the local optimizer is invoked.

### 3.6 Multilevel DIRECT (MrDIRECT and MrDIRECT$_{075}$)

Liu et al. [38,39] address the weakness of global drag and slow refinement of local optima with an idea that, in spirit, is the same as the revised DIRECT by Jones discussed in the previous section. Namely, they use a local optimizer to fine tune solutions. The innovation is that, instead of using some other algorithm as the local optimizer, they use DIRECT itself, with some modifications, to do the local search.

More specifically, in their MrDIRECT$_{075}$ algorithm, DIRECT is run at three levels called 2, 1, and 0:

- At level 2, DIRECT is run as usual, with $\epsilon = 10^{-5}$.
- At level 1, the rectangles from level 2 are sorted first by size and then function value and rectangle selection is limited to the first 90% of the rectangles; that is, the biggest 10% of the rectangles are ignored. A value of $\epsilon = 10^{-7}$ is used.
- At level 0, the rectangles from level 1 are sorted in the same way and only the first 10% are considered for selection. A value of $\epsilon = 0$ is used. Thus, this level focuses on local refinement.

[4] https://pypi.org/project/Py-BOBYQA/.

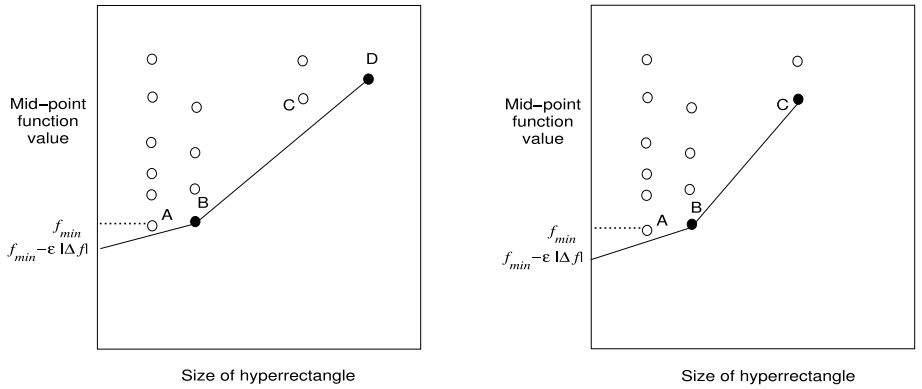


**Fig. 23** Left: hypothetical set of rectangles in the partition at level 2; rectangles B and D are selected. Right: at level 1, the biggest 10% of the rectangles are ignored, so rectangle D is not considered, and hence rectangles B and C are selected. Thus, using level 1, we can select some rectangles (C in this case) that would have otherwise been dominated by the biggest rectangles. (Reproduced from Liu et al. [39])

The algorithm cycles through the levels according to the following pattern, called the "W-cycle": 21011012 (this pattern emerges from a recursive calling structure). By ignoring most of the bigger hyperrectangles at level 0, and by using $\epsilon = 0$, the algorithm can refine the solution to high accuracy with little global drag. Due to the positive values of $\epsilon$ at the other levels, using $\epsilon = 0$ at level 0 does not lead to an excessive amount of local search.

Another advantage of the approach stems from ignoring the biggest 10% of the rectangles at level 1. This can lead to the selection of some intermediate-sized rectangles that would otherwise have been dominated by the (now ignored) biggest rectangles, thereby making the global search more thorough (recall that insufficient global search is one of the potential weaknesses of DIRECT). How this works is detailed in Fig. 23. On the left of the figure, we show the selection diagram for a hypothetical set of rectangles at level 2, where all the rectangles are considered but only rectangles B and D are selected. On the right, we assume we are at level 1 and have ignored the biggest 10% of the rectangles, so that rectangle D is ignored. In this case, rectangles B and C are selected. This shows how ignoring some of the bigger rectangles at level 1 can lead to selecting some rectangles (C, in this case) that would otherwise have been dominated by the biggest rectangles.

Finally, by returning to level 2 every so often and considering all the rectangles, they insure that full global search is still carried out. Their original MrDIRECT algorithm is the same as MrDIRECT$_{075}$ except that they use $\epsilon = 10^{-4}$ at all levels. Numerical results showed that varying the value of $\epsilon$ at the different levels improved performance, especially when using $\epsilon = 0$ in level 0.

Liu et al. [38] compare MrDIRECT and DIRECT using the GKLS [19] software, which can generate global optimization test problems with different dimensions and difficulty. On problems varying from 2 to 20 variables, using a computational budget of $1000(n + 1)$ evaluations, they find that MrDIRECT is more efficient, solving 72% of the problems compared to 36% for DIRECT. Most interestingly, they find that MrDIRECT reduces the curse of dimensionality, being able to solve 61% of the 20-variable problems, compared to only 11% for DIRECT.

In a later paper, Liu et al. [39] do a thorough comparison on the Hedar test set [25] between MrDIRECT$_{075}$, DIRECT, and the original MrDIRECT. The Hedar test set consists of 68 problems with up to 48 design variables. They find that MrDIRECT$_{075}$ can solve 65%

of the problems with the fewest function evaluations, compared to only 20% for DIRECT. With a computational budget of up to $5000(n+1)$ function evaluations, MrDIRECT$_{075}$ is able to solve 85% of the problems with a relative accuracy of $10^{-6}$, whereas DIRECT can only solve 63% of the problems with that accuracy.

Overall, the MrDIRECT$_{075}$ algorithm addresses many of the weaknesses of DIRECT we identified earlier. Global drag is reduced by ignoring the bigger rectangles in levels 1 and 0 of the algorithm, and global search is also made more thorough by the use of level 1 which, as explained in Fig. 23, can lead to some intermediate sized rectangles being selected that might otherwise be overlooked. On the other hand, MrDIRECT$_{075}$ does nothing to address the weakness of ignoring obvious trends in the data, such as monotonicity; the authors deliberately chose to avoid using a separate local optimizer (which could exploit trends), since they wanted to improve performance while staying within the DIRECT's space-partitioning paradigm.

### 3.7 Adaptive diagonal curves (ADC)

Sergeyev and Kvasov [63] introduced a version of DIRECT with two key modifications: (1) a change to the partitioning scheme that uses the concept of "adaptive diagonal curves" [33,62] and, (2), a change to the rectangle selection that avoids excessive search around suboptimal local minima and promotes a sufficient level of global search.

Let us start with their modification of the partitioning scheme. Recall that, in DIRECT, each rectangle is evaluated based on just *one* sampled point—the center point. As a result, a problem can be deceptive if the global minimum lies in a rectangle whose centerpoint has a bad function value (as is the case with the Shubert function; see Fig. 13). To reduce this possible deception, Sergeyev and Kvasov sample *two* points in each rectangle, where the two points are opposite vertices. i.e., located on a main diagonal. For example, the left panel of Fig. 24a shows the unit hypercube sampled at opposite points $a = (0, 0)$ and $b = (1, 1)$. This diagonal sampling makes the evaluation of each rectangle not dependent on a single point, so we could expect its performance to be less susceptible to deceptive problems. On the other hand, one might think that this sampling procedure would double the number of required function evaluations, since each rectangle is sampled at two points. Surprisingly, this turns out *not* to be the case: The new scheme *reduces* the number of function evaluations required to sample a given number of rectangles. To see, why, let us again consider in Fig. 24, where we show three iterations of Sergeyev–Kvasov algorithm.

At the start of the search, as shown in Fig. 24a, there is just one rectangle—the unit hypercube—sampled at the diagonal points $a = (0, 0)$ and $b = (1, 1)$. So at this stage, the ratio of sampled points to rectangles is indeed $2 : 1$. Since this is the only rectangle, it is the one that is selected.

Sergeyev and Kvasov [63] trisect selected rectangles along just *one* of the long sides. In Fig. 24a, we trisect along the horizontal dimension, sampling at the new points $u = (\frac{2}{3}, 0)$ and $v = (\frac{1}{3}, 1)$. This splits the rectangle into thirds and replicates the property that each resulting child rectangle is sampled at two opposite vertices.

In Fig. 24b, the shaded rectangle is selected and then subdivided in the same way. Finally two rectangles are selected and subdivided (Fig. 24c). We now have nine rectangles and nine sampled points, so the ratio of points to rectangles becomes $1 : 1$. The reason why this is possible is that *many rectangles share vertices*. As the search continues, the number of sampled points can tend to be many fewer than the number of rectangles; Sergeyev and

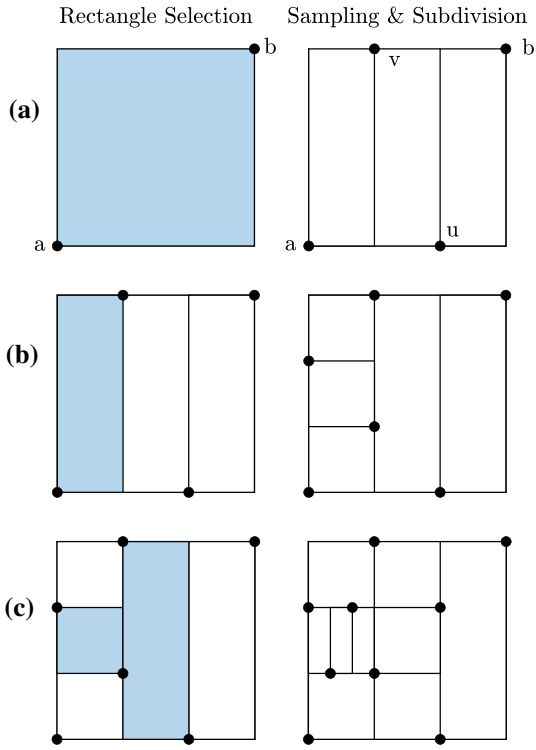


**Fig. 24** Three iterations using the trisection and diagonal sampling method of Sergeyev and Kvasov [63]

Kvasov [63] cite one case in which there are 106,359 hyperrectangles but only 15,343 sampled points.

The second innovation is on rectangle selection. The authors were primarily concerned about DIRECT's potential to spend too many function evaluations in local search around a suboptimal local, with insufficient global search, thereby delaying the discovery of the global minimum for a long time. This weakness is most apparent with very hard multimodal problems, and the authors were primarily concerned with improving DIRECT's performance for such hard problems. To address this weakness, they introduce distinct "local" and "global" phases to rectangle selection, making sure not to spend too many iterations in the local phase and, likewise, making sure to spend enough iterations searching globally.

Without going into too much detail, we can explain the gist of their approach as follows. Rectangle selection is done with the aid of a diagram similar to that used by DIRECT (see Fig. 7), except that on the vertical axis they replace the centerpoint function value with the average function value at the two opposite vertices. The difference between the local and global phases of selection lies in the range of rectangle sizes that are considered.

In the local phase, rectangles of all sizes are considered, and $n+1$ iterations are performed. Thus the local phase is more or less like the usual DIRECT selection approach (we are leaving out some minor details). If the current best solution improves by at least 1%, local search is deemed to "be working" and so another $n+1$ local iterations are performed. Local search is also continued if the current best point does not lie in one of the smallest rectangles (since we might find an even better local solution by further subdividing this rectangle). However, if we do *not* get the desired 1% improvement, and if the rectangle with the current best point

*is* one of the smallest rectangles, then we consider local search to be complete and we switch to the global phase.

In the global phase, we limit selection to the "bigger" rectangles to make the search more global and hopefully find the basin of a better local optimizer. To make the notion of "bigger" more precise, let $q$ denote the number of divisions of the biggest rectangle in the partition and $p$ denote the number of divisions of the rectangle with the current best point (we have $p \geq q$). To limit selection to the "bigger" rectangles, we only consider rectangles that have been divided no more than $r'$ times, where $r' = \lceil (q + p)/2 \rceil$. For example, suppose that the biggest rectangle has been trisected $q = 10$ times, and that the rectangle with the best sampled point is smaller, having been trisected $p = 15$ times. In the global phase, we would only consider for selection rectangles that had been trisected no more than $r' = \lceil (q + p)/2 \rceil = 13$ times. In this way, we ignore the smaller rectangles that have been divided 14 or 15 times. These global iterations that focus on the bigger rectangles are continued until we either improve upon the solution from the previous local phase by 1% or until $2^{n+1}$ global iterations have been performed.

Comparing the ADC algorithm to MrDIRECT, we see it is almost the opposite. Whereas MrDIRECT had one phase that was like the usual DIRECT and two other phases in which the search focused on *smaller* rectangles, the ADC approach has a local phase that is more or less like the usual DIRECT and a global phase that focuses on *bigger* rectangles.

Unfortunately, the papers for the two algorithms do not report results on the same test problems in a way that would enable us to discern which approach works best. What is clear is that on the easy Dixon–Szego test problems used in the original DIRECT paper [30], MrDIRECT does better than DIRECT, while the Sergeyev–Kvasov approach does worse. However, on the most difficult 4 and 5 dimensional problems from the GKLS test suite, ADC is able to find solutions where DIRECT fails. On these harder problems, the emphasis on global search in the ADC algorithm pays off.

In a later study, Sergeyev et al. [65] performed a thorough comparison of ADC, DIRECT, and DIRECT-L on easy and hard problems of 2 to 5 dimensions from their GKLS test suite. They find that for easy problems, the original DIRECT algorithm was able to solve more problems than ADC or DIRECT-l when limited to a computational budget of no more than 14,000 runs. However, for hard problems, or for easy problems with a high computational budget, they find that ADC solves more of the problems. Interestingly, in all their examples, DIRECT does better than DIRECT-l, suggesting that DIRECT-l only performs best on *very* easy problems, such as the Dixon–Szego test problems used in the original DIRECT article [30]. The bottom line is that the extra emphasis on global search in ADC has the expected effects: it improves performance on hard problems where more global search is needed, and it degrades performance on easier problems.

One thing is definitely clear: The diagonal sampling approach in ADC is delivering a great deal of value regardless of the problem difficulty. For example, on their hardest problem, they finish with 685,173 rectangles but sample only 77,981 points.

It would be interesting to know the separate impact of diagonal sampling; that is, how would the original DIRECT algorithm (or variants like MrDIRECT) perform if we kept everything the same but replaced centerpoint sampling with diagonal sampling? One hint as to what might be gained comes from the work of Paulavičius and Žilinskas [50] on *simplex*-based DIRECT. They implement a version of DIRECT, called DISIMPL-V, in which the search space is divided into simplices, each of which is sampled at its $n + 1$ vertices. The selection step is in spirit exactly the same as DIRECT, making some fairly straightforward changes to account for the move from rectangles to simplices. Because neighboring simplices can share vertices, it turns out that the number of sampled points can be far less than the

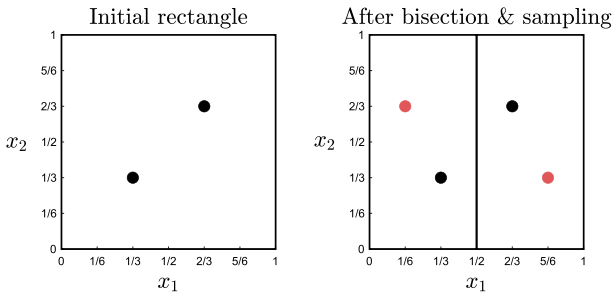


**Fig. 25** Rectangle bisection and sampling as implemented in the BIRECT algorithm

number of simplices—similar to how the number of sampled points can be far less than the number of rectangles in diagonal sampling. In their numerical work, they find that DISIMPL-V is sometimes better and sometimes worse than DIRECT, with a slight edge overall for DISMPL-V. If this is any indication, we could expect some overall improvement if diagonal sampling were used in rectangle-based DIRECT, MrDIRECT, or DIRECT-GL.

In a later paper, Paulavičius et al. [51] extend these simplex-based DIRECT variants to use the same two-phase, globally-biased selection introduced in the rectangle-based ADC algorithm. The results were very much the same (namely, the globally-based selection improves performance on hard problems).

### 3.8 DIRECT using bisection of rectangles (BIRECT, BIRECT-l, Gb-BIRECT-l)

Paulavičius et al. [52] have developed an innovative variant of DIRECT called BIRECT that samples two points per rectangle and, furthermore, employs rectangle bisection instead of trisection. The key idea is shown in Fig. 25. On the left, the initial rectangle is sampled at two points located $1/3$ and $2/3$ of the way along the main diagonal. On the right, the rectangle has been bisected along the horizontal ($x_1$) axis. Each child rectangle inherits one sampled point from the parent (shown in black) and a new point is also sampled (shown as red) in such a way that we maintain the property that every rectangle has two sampled points located $1/3$ and $2/3$ along a diagonal. The diagonal along which the two points are located may change, as in does in Fig. 25, but Paulavičius et al. [52] show that this scheme works not only in this simple example but also more generally, for any number of dimensions. Bisection is always applied to one of the long sides of a rectangle and, if there is a tie for longest, they select the side with the lower index (as we did in Fig. 25).

Rectangles are selected in essentially the same way as DIRECT, using our usual "function value vs rectangle size" diagram (see Fig. 7), with the change that the vertical axis has the *minimum* of the function values at the two sampled points, and the horizontal axis is $2/3$ the length of the diagonal (as opposed to $1/2$ as in DIRECT).

What is gained by this innovation? Well, recall that one reason that DIRECT may converge slowly is that the rectangle containing the optimum has a center point with a very bad function value. This bad function value delays the selection of the rectangle for further search because other rectangles of the same size or larger that have better function values need to be selected first. By sampling *two* points in each rectangle, BIRECT reduces the likelihood that the rectangle with the global minimum has a bad function value in the selection diagram since, for this to happen, we would require not just one, but two points to "unluckily" align with areas where the function value is bad.

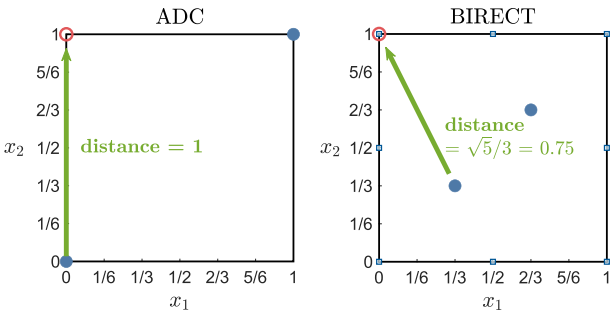


**Fig. 26** The maximum distance between the sampled points (solid blue circles) and an unsampled point (open red circle) is higher in the ADC method than in the BIRECT method. (Color figure online)

Now in the previous section we described the adaptive diagonal curve (ADC) method that also samples two points per rectangle, and so ADC also has the advantage just mentioned. But with ADC the two points are *corner* points of a diagonal, not interior points. The method used in BIRECT can be argued to be superior, for two reasons.

First, as shown in Fig. 26, the maximum distance between an unsampled point in a rectangle and the closest of the two sampled points is lower in BIRECT than in ADC. As a result, for any given Lipschitz constant, and assuming the function values at the two points are the same, the Lipschitzian lower bound will be tighter when using BIRECT. Put another way, the sampled points in BIRECT give a better indication of what other function values are likely to be found in the rectangle than do the corner points in ADC.

The second advantage of BIRECT is its use of bisection instead of trisection, which implies that the aspect ratio of rectangles is never more than 2:1 as opposed compared to 3:1 for ADC and DIRECT. Given two rectangles with the same volume, but different aspect ratios, the rectangle with the lower aspect ratio has the smaller center-to-vertex distance, and so the tighter Lipschitzian lower bound. Intuitively, the lower the aspect ratio, the easier it is for one or two sampled points to be "representative" of what the objective function might be at other points in the rectangle.

All the above arguments suggest that BIRECT would be more effective than ADC. On the other hand, BIRECT samples exactly two points per rectangle whereas, as we discussed in the last section, with the ADC method the ratio of sampled points to rectangles can be much less than two due to the shared vertices. One might think that this lower ratio of sampled points to rectangles would give an advantage to ADC. The net effect of these advantages and disadvantages is, as of this writing, unclear.

In a later paper, Paulavičius et al. [53] extend the BIRECT algorithm to use the same "globally biased," two-phase selection process as in their ADC algorithm, creating an algorithm they call Gb-BIRECT. Paulavičius et al. [53] also develop a version of BIRECT called BIRECT-l which differs from BIRECT in that, if several rectangles are tied for being potentially optimal, only one of them is selected. As we saw with the original DIRECT, this change generally improves convergence. The same change was made to Gb-BIRECT creating a version called Gb-BIRECT-l.

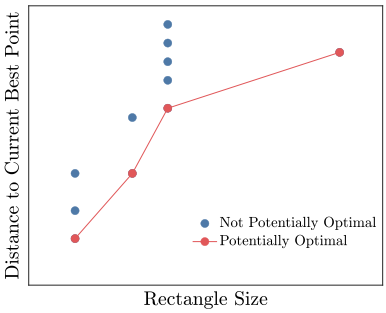


**Fig. 27** Example of rectangle selection with "local enhancement" in the DIRECT-GL algorithm

### 3.9 Two-step, global-local selection (DIRECT-GL)

As we commented in the discussion of Fig. 14, there are two reasons why DIRECT may take a long time to find the global minimum. First, it may spend an excessive number of function evaluations exploring a suboptimal local minimum, that is, the search is too local. Second, the global search may be insufficiently thorough.

Stripinis et al. [68] make the global search more thorough via the same technique mentioned in Sect. 3.4 when discussing the PLO and PLOR algorithms: select all rectangles that are nondominated (Pareto optimal) with respect to rectangle size and centerpoint function value (see Fig. 21b). The authors call this the "global selection step." However, as mentioned earlier, the increased amount of global search causes more "global drag" and slows down the refinement of solutions to high accuracy. To counteract this effect, the authors again want to do local search but, as in MrDIRECT, they want to stay within DIRECT's space-partitioning paradigm. What they do, therefore, is to add a second, "local selection step." This local selection step uses our usual selection diagram, except that the vertical axis now has the *distance from each rectangle's center to the current best point* instead of the centerpoint function value (see Fig. 27). The horizontal axis has rectangle size as before. They then select the rectangles on the lower-right efficient frontier, those that are nondominated with respect to distance from the current best point (lower is better) and rectangle size (higher is better). Intuitively, they want to focus on rectangles that are close to the current best point (so as to do local search), and among those that are close, they prefer bigger rectangles because there is more unexplored space.

Putting it all together they now do a two-step selection process in each iteration. First, in the "global step," they select rectangles that are nondominated with respect to centerpoint function value versus rectangle size, as in Fig. 21b. These rectangles are then fully processed (subdivided and sampled). Second, in the "local" step, they select rectangles that are nondominated with respect to distance to the current best point and rectangle size, as in Fig. 27. These rectangles are then fully processed.

The authors do extensive numerical comparisons of DIRECT and DIRECT-GL on the Hedar test set. They find that the original DIRECT is faster for unimodal test functions. Intuitively, for a unimodal function, there is no need to worry about getting stuck on a suboptimal local or doing insufficient global search because there is only one local minimum. However, on multimodal problems, DIRECT-GL converged in significantly fewer function evaluations on average; moreover, relative to DIRECT, asking for higher accuracy incurs a lower cost in additional function evaluations (this is probably due to the special local selection step).

Overall, the DIRECT-GL algorithm addresses most of the key weaknesses of DIRECT, with the exception of not recognizing or exploiting any obvious trends in the objective function. Another seeming advantage is the elimination of the $\epsilon$ parameter, yielding an algorithm with absolutely no hyperparameters. However, with the elimination of $\epsilon$, the algorithm loses any mechanism that stops it from selecting very small rectangles, so we might think that this would lead to poor performance on the Shubert problem, just as it did for the original DIRECT algorithm. In fact, if they *only* use global selection step (what they call DIRECT-G), they do take longer to converge to $10^{-2}$ accuracy when compared to regular DIRECT (4089 evaluations vs. 2967). However, when they use both local and global enhancements, they converge to $10^{-2}$ accuracy on the Shubert problem in only 425 evaluations. For the Shubert problem, the global minimum is close to the local, so the fact that the local step of selection focuses on rectangles close to the current best point probably explains why DIRECT-GL is able to find the global so quickly in this case. However, DIRECT-GL does not always perform so well; on the Hartman 6 problem, it takes 8793 function evaluations, versus 571 for DIRECT.

Now let us compare the DIRECT-GL method with the globally-biased, two-phase approaches used in ADC and Gb-BIRECT. In both cases there are two phases to rectangle selection. Moreover, in both cases, the first phase emphasizes global search and the second phase emphasizes local search. The difference between the methods lies in *how* the global or local emphasis is achieved. The ADC and Gb-BIRECT algorithms emphasize global search by focusing on bigger rectangles, whereas DIRECT-GL emphasizes global search by selecting rectangles that are Pareto optimal (which is a potentially larger set of rectangles than those on the convex hull in the selection diagram). Likewise, ADC and Gb-BIRECT algorithm emphasize local search by simply removing the emphasis on larger rectangles and running selection as "usual," whereas DIRECT-GL puts a more specific emphasis on local search by favoring rectangles whose center points are close to the current best point.

### 3.10 DIRMIN

In Sect. 3.5, we discussed how Jones [29] suggested hybridizing DIRECT with a separate local optimizer, initially running the local optimizer from the current best point after a fixed number of function evaluations (e.g., 100) and then running it again whenever the global search finds a point that improves upon the last local solution. Liuzzi et al. [40] take this idea one step further and suggest running a local search starting from the centerpoint of *every* potentially optimal rectangle. They call this variant DIRMIN. The local search method they use is a truncated Newton method.

This is a simple idea that solves all the problems in the Hedar test set except for three challenging problems of dimensionality 10, 25, and 50. However, it is likely that their approach generates more local searches than are really necessary, as many of the starting points probably converge redundantly to the same local optimum. For example, on the Shubert problem, where the Jones' approach in Sect. 3.5 required 6 local searches, Liuzzi et al. [40] require 27 searches.

Liuzzi et al. [40] also introduce another variant, called DIRMIN-TL, to address the three problems that DIRMIN failed to solve. In particular, suppose that DIRMIN reaches its iteration limit (or memory limit) and has to stop with the current best point being $\tilde{x}$. They then suggest transforming the problem so that $\tilde{x}$ maps to centroid of the unit hypercube. This is achieved using the piecewise linear transformation $y = T_{\tilde{x}}(x)$ shown in Fig. 28 for the $i$th variable, which maps $\tilde{x}_i$ to 0.5. The search is then done over the transformed $y$ variables with

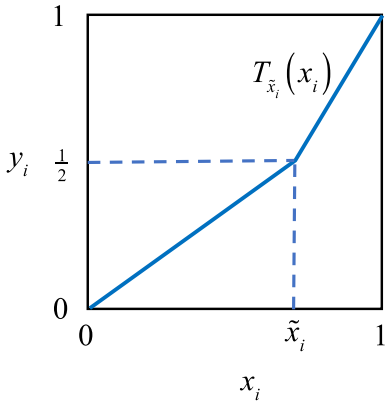


**Fig. 28** Transformation of variables used in DIRMIN

the objective being $f_{\tilde{x}}(y) = f(T_{\tilde{x}}^{-1}(y))$. With this transformation, the first point in the new DIRECT search, the centroid in the $y$ space, corresponds to $\tilde{x}$ in the original space. In this way the new search focuses around the previous best point, and the hope is that this fresh start finds the global minimum before it gets stuck in a suboptimal local minimum or slowed down by "global drag" stemming from a partition with many rectangles.

This whole process—solving with DIRMIN and then restarting centering on the new solution—is then repeated a number of times. With DIRMIN-TL, using up to 26 restarts, the authors are able to solve the three problems that could not be solved with DIRMIN itself. In a later paper, Liuzzi et al. [41] extend the approach, hybridizing DIRECT with a derivative-free local optimizer and obtaining similar results.

Overall, this method has the advantage that, by using a local search algorithm, it follows the obvious trends in the objective function. However, the number of local searches is likely excessive. Moreover, there is no strong theoretical reason to believe that restarting the entire search with a space transformation should yield improvement. The main motivation for the restarts seems to be that, in high dimensions, DIRECT may run out of memory before finding the optimum. In this case there is no way to continue DIRECT, so they restart.

### 3.11 glcCluster

glcCluster is a DIRECT-inspired search procedure that was one of the most successful in the comparison of derivative-free search algorithms carried out by Rios and Sahinidis [57]. It did especially well with respect to nonconvex, smooth problems. The algorithm is available as part of the TOMLAB suite of optimization tools for MATLAB, and a description of the method is available on the TOMLAB website.[5] According to this description, glcCluster is a hybrid algorithm, combining DIRECT, a clustering algorithm, and local search. It takes advantage of the previously mentioned tendency for the sampled points from DIRECT to cluster (be more dense) in high-performing regions of the space, as shown in Fig. 9.
The algorithm has five steps:

1. Run DIRECT (the updated version [29]) for a suitable number of function evaluations (default is $100n + 1$ where $n$ is number of variables).
2. Run an adaptive clustering algorithm on the sampled points to find a suitable number of clusters.

[5] http://tomwiki.com/GlcCluster#Description/.

3. Run a local search from the best point in each cluster. The local solver is either SNOPT or NPSOL, using analytical gradients if provided, or finite-differences otherwise.
4. Run DIRECT again, warm starting from where we stopped in Step 1 and using the updated value of $f_{\min}$ from the local searches. The goal is to improve upon the previous solutions from local optimization.
5. If the run of DIRECT in Step 4 improves the best point, a local search is finally made from this best point.

Intuitively, glcCluster acts much like Jones' hybrid of DIRECT and local search, with the difference that, instead of running local search just from the current best point, glcCluster uses a clustering algorithm to find *several* good starting points. Local search is then done from these points. The excellent performance of glcCluster in the Rios–Sahinidis comparison of derivative-free algorithms is probably due the ability of the clustering step to identify promising regions for local search, and to do this much more economically than DIRMIN (which launches local searches from the center of every potentially optimal rectangle).

### 3.12 DIRECT with bisection, globally-biased selection, and local search (Gb-BIRMIN)

In Sect. 3.8 we discussed the Gb-BIRECT version of DIRECT introduced by Paulavičius et al. [53] that used bisection instead of trisection and a special "globally-biased" variant of rectangle selection. In general, this method excels on the *hard* test problems where more global search was needed. However, for *simple* problems, what is needed is a way to quickly refine a solution to high accuracy, and both DIRECT and Gb-BIRECT are weak in this regard. The convergence speed was somewhat improved by making Gb-BIRECT only select one of any set of rectangles that were tied for being potentially optimal, a version they called Gb-BIRECT-l. To improve local refinement further, Paulavičius et al. [53] enhance Gb-BIRECT-l by hybridizing it with a local optimizer (MATLAB's "fmincon"). Similar to Jones' revision of DIRECT discussed in Sect. 3.5, the local optimizer is first launched from the current best point after the feasible region is "at least slightly explored." After the local optimizer is complete, global search is resumed as in Gb-BIRECT-l, but the search now evolves differently because the current best solution $f_{\min}$ has (most likely) been improved. Local search is restarted whenever the current best solution from Gb-BIRECT-l improves upon the previous best solution. In this way, the algorithm alternates between global and local search until it reaches a limit on total function evaluations. The authors call this hybrid algorithm Gb-BIRMIN.

A unique aspect of Gb-BIRMIN is its way of deciding when to invoke the local optimizer. Paulavičius et al. [53] consider several possibilities. Like Jones [29], one option they consider is obvious: to launch the local optimizer as soon as global search finds a point that is better than the last local solution by some non-trivial amount (they use $0.01|f_{\min}|$). Interestingly, they find it best not to start the local optimizer immediately after global search improves upon the last local solution, but rather let the global search keep going as long as it continues to make improvements, and only launch the local optimizer when one iteration of global search results in no further improvement. The option that worked the best, however, was to start the local optimizer whenever the global search with Gb-BIRECT-l found a rectangle center point with a function value that was non-trivially better than the *last best point found by Gb-BIRECT-l*, that is, the best point not counting those found in local search. This third option launches the local optimizer more often, since it not only launches the local optimizer when global search finds a point that is better than the last local solution, but also launches it when global search finds a point non-trivially better than what was previously found by

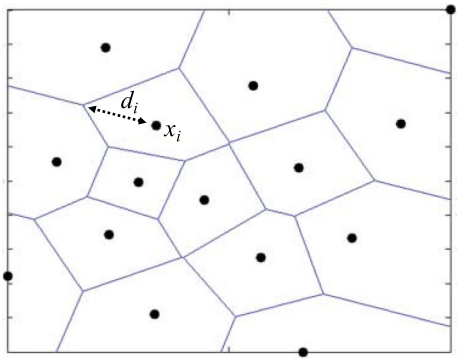


**Fig. 29** Sampled points and corresponding Voronoi cells. The size of the cell $i$ is captured by the distance $d_i$ between the cell center $x_i$ and the furthest vertex of the Voronoi cell. Reproduced from Liu et al. [36]

global search alone. These different versions of Gb-BIRMIN were compared to the previously discussed DIRMIN using the same test problems used in Liuzzi et al. [41]. They found that Gb-BIRMIN—using the third option for launching the local—performed significantly better than DIRMIN, requiring an average of 86,494 evaluations over the test problems, versus 183,567 for DIRMIN.

In our view, Gb-BIRMIN is one of the most promising and sophisticated enhancements yet made to DIRECT. The only weakness of DIRECT that it does not explicitly address is DIRECT's inability to recognize clear trends between the sampled points—that is, both DIRECT and Gb-BIRMIN do not extrapolate and interpolate obvious trends in the data in the way that could be done using, for example, a fitted kriging surface or radial-basis functions. We consider an algorithm that does interpolate and extrapolate trends in the next section.

### 3.13 Voronoi-based partitioning with metasurface (eDIRECT)

The eDIRECT algorithm of Liu et al. [36] is an extension of DIRECT that allows for a very flexible partitioning scheme. The algorithm can start with any set of $m$ points, such as a Latin hypercube. It then uses these points to divide the space into $m$ regions called "Voronoi cells." For sampled point $i$, the associated Voronoi cell is the set of all points that are closer to point $i$ than they are to any other point $j \neq i$. An example is shown in Fig. 29. The size of a Voronoi cell can be taken to be the greatest distance between the sampled point and a vertex of the cell. In Fig. 29, we label one sampled point $x_i$ and show the associated cell size $d_i$.

Given that we have the function value $f(x_i)$ at the center of cell $i$ and the cell size $d_i$, we can make our usual selection diagram and select "potentially optimal Voronoi cells" by finding the lower-right convex hull of the points in plot of cell function value versus cell size.

To complete the algorithm, all that remains is to decide where to sample within each selected Voronoi cell. One possibility is to sample the point that is furthest from the center, and the authors do this for the biggest selected Voronoi cell, to assure that all Voronoi cells get smaller over the search. For the other selected cells, however, eDIRECT does something more involved. In particular, it fits a response surface, usually a kriging surface, to the observed data. Having fit the response surface, it might seem natural to sample the point in the Voronoi cell that minimizes the surface. But trusting the surface in this way only makes sense if the surface is accurate; if the surface were inaccurate, it would be better to sample at a point most distant from the Voronoi cell center, so as to better fill the space and improve the surface fit. To balance these two concerns, the authors find a point in the cell that optimizes a weighted sum of distance from the cell center (higher is better) and predicted function value (lower

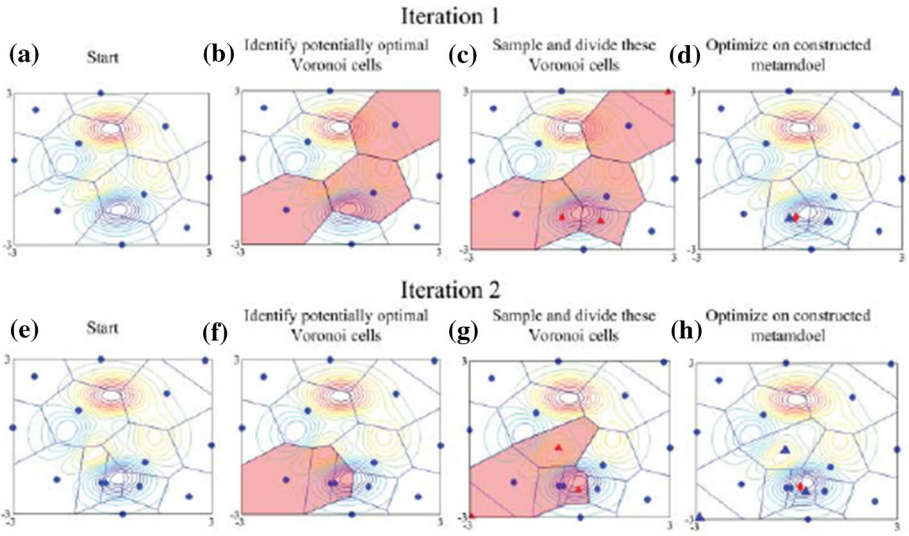


**Fig. 30** Two iterations of eDIRECT. The contours in the figure are for the objective function (which, of course, would not be known in a real problem); the global minimum is in the lower center of the plot. Reproduced from Liu et al. [36]

is better). The weight used in this criterion takes into account the accuracy of the surface as indicated by cross-validation; the more accurate the surface in a region is (low cross-validated error), the more weight is given to sampling where the predicted function value is low.

After sampling within the selected Voronoi cells, eDIRECT adds a final step: it finds the global minimum of the response surface and samples that point as well. This step is similar to running a local optimizer in DIRMIN or BIRMIN, except that local optimization is done on the response surface, not the function itself. Unlike DIRMIN and BIRMIN, however, the resulting local solution can be added to the partition due to the flexibility of the Voronoi approach. Figure 30 illustrates the steps of the algorithm, showing two iterations of eDIRECT on a hypothetical two-variable problem.

Recall that there are two steps in eDIRECT that involve optimization over a Voronoi cell:

1. Calculate the size of a Voronoi cells. This involves maximizing the distance from the Voronoi center, subject to being within the Voronoi cell.
2. Decide where to sample within the cell. This involves optimizing a weighted sum of distance from the cell center (higher better) and predicted function value (lower better).

These optimizations are solved approximately as follows. A large set of $50 \times m \times n$ random points is generated in the feasible region, where $m$ is the number of points sampled so far in the search, and $n$ is the number of input variables. For each point $i$, the subset of random points that lie in Voronoi cell $i$ is identified (i.e., the random points that are closest to point $i$). The two optimizations are then solved by enumeration over these candidate points. Thus, the explicit boundaries of the Voronoi cells are not computed—a good thing, since this is computationally intensive in more than six dimensions.

In their numerical results, Liu et al. [36] compare the performance of eDIRECT on test functions with 2 to 10 dimensions against DIRECT and two other metamodel-based optimization algorithms. On all these problems, they significantly outperform the competitors. For example, on the six-dimensional Hartman 6 function, they converge on average in 120

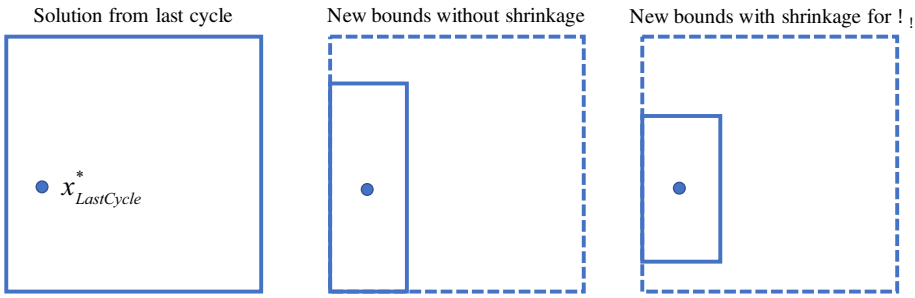


**Fig. 31** Redefining variable lower and upper bounds between cycles of DIRECT

evaluations, whereas DIRECT takes 571. They also study a version of the Hartman 6 problem in which the bounds are perturbed so that the minimum is on the boundary. As expected, DIRECT takes longer when the solution is on the boundary, requiring 2449 evaluations; in contrast, eDIRECT requires only 125.

Compared to all the other modifications of DIRECT, eDIRECT goes the furthest in addressing the inflexibility of DIRECT's partitioning scheme. In fact, eDIRECT's partitioning scheme is almost infinitely flexible, since the Voronoi approach can accept any set of points. By fitting a response surface to the sampled points and then sampling the point that minimizes this surface, eDIRECT is able to exploit clear trends in the sampled points and refine solutions quickly.

One might think that the algorithm would be sensitive to the number of random points used to approximate the Voronoi regions, but a sensitivity study indicates that using $50 \times m \times n$ points is sufficient; using more (e.g., $100 \times m \times n$) does not help much and sometimes actually degrades performance a little on their test problems.

### 3.14 Zoom-in strategy for high-dimensional problems (HD-DIRECT)

Tavassoli et al. [73] explore a "zoom-in" strategy to help DIRECT perform better on high-dimensional problems. Instead of a single run of DIRECT, the algorithm is run for several "cycles." In the first cycle, DIRECT is run with the original lower and upper bounds on the variables, stopping when there is little change in the best function value over the last several iterations, indicating that progress has slowed. Another cycle of DIRECT is then started, but this time the lower and upper bounds are redefined so that the best point from the last cycle is at the center of the design space. This re-centering is illustrated in Fig. 31, where we can see that the solution from the last iteration ($x^*_{LastCycle}$) is closer to the lower bound for $x_1$ than it is to the upper bound. The upper bound is then adjusted so that $x^*_{LastCycle}$ is in the middle of the bounds. The upper bound for $x_2$ is similarly adjusted so that $x^*_{LastCycle}$ is in the middle of the bounds. The figure also shows how a further shrinkage of the range is made when a variable ($x_2$ in this example) has changed little over the final 10 iterations of DIRECT in the last cycle.

When restarting a new cycle of DIRECT, all previous function evaluations are ignored. What changes are the lower and upper bounds on each variable. Because DIRECT is restarted, there is less global drag slowing down local refinement of the incumbent solution, so local refinement can be expected to proceed faster.

Of course, any zoom-in strategy runs the risk of zooming in on a local minimum and excluding the region that contains the global minimum. To prevent this, the authors add a "diversification subroutine" that randomly runs with a small probability (they use 0.1) during each cycle. The diversification subroutine evaluates random points in regions that have been excluded due to the zooming in (they use $10n$ points), picks the one with the minimum objective value, and starts a new cycle of DIRECT centered about this point. If a better answer is found, the zoom-in strategy jumps to this solution in the next iteration.

The algorithm stops when either: (1) the optimum function value changes very little between two cycles of DIRECT, (2) all $n$ variables show a little change in the last $n$ iterations of DIRECT, or (3) the maximum allowed number of cycles is reached.

The authors call this high-dimensional version of DIRECT by the name HD-DIRECT and compare its performance to the original DIRECT for nine test functions with 15 to 30 variables, as well as one 30-variable industrial application. In these numerical experiments, HD-DIRECT converges in many fewer function evaluations than DIRECT. For example, for the sphere problem $f(x) = \sum_1^n x_k^2$, with bounds $[-3, 7]^n$ and $n = 15$ variables, the original DIRECT requires over 1,000,000 evaluations to get to a solution within 0.01 of the optimum (zero), whereas HD-DIRECT requires 22,562 evaluations. The savings are even more dramatic when higher solution accuracy is desired. Even so, 22,562 function evaluations is excessive for such a simple convex problem! Any extension of DIRECT that incorporates some trajectory-following local optimization algorithm would do much better. For example, the eDIRECT algorithm, which fits a metamodel and optimizes it, is able to solve a 10-dimensional sphere problem in only 293 evaluations [36]. As another example, HD-DIRECT takes 29,660 evaluations to optimize the linear objective $f(x) = \sum_1^n x_k$, on $[0, 5]^n$ with $n = 30$; for such a linear function, any method hybridizing DIRECT with a gradient-based local optimizer would find the solution on the first local search.

### 3.15 Other extensions of DIRECT

Up to this point, we have discussed extensions of the original DIRECT algorithm with only bound constraints on the variables. In fact, our focus here has been to obtain a deeper understanding of the weaknesses of this original version of DIRECT and to survey approaches in the literature to overcome them. However, several authors have explored extensions of DIRECT that enable it to handle nonlinear constraints, multi-objective problems, problems with noisy objective functions, and problems with integer variables. For completeness, we briefly mention some of these extensions in this section.

Jones [29] extended DIRECT to handle nonlinear inequality constraints by forming a special auxiliary function that heuristically captures the likelihood that a rectangle contains the global (feasible) optimum with function value $f^*$. Of course, we do not know $f^*$, so the algorithm selects any rectangle that can have the best value of the auxiliary function for *some* $f^* \leq f_{\min} - \epsilon |f_{\min}|$. Values of $f^*$ close to $f_{\min}$ select rectangles good for local search, while values of $f^*$ approaching $-\infty$ select rectangles good for global search. When there are no constraints, this approach reduces to the original method for rectangle selection and can hence be considered a natural generalization. To constrain some variables to take on integer values, the rectangle subdivision procedure was modified so that the centerpoint always has an integer value for any integer variable.

In their eDIRECT algorithm, Liu et al. [37] add nonlinear constraints by modifying how the regions (Voronoi cells) are selected for sampling and subdivision. If no feasible point has been found, they select regions using the lower-right convex hull in a diagram of normalized

constraint violations versus region size. Once a feasible point has been found, they then do the selection in two steps. First, they do the selection considering only the infeasible regions (based on constraint violation vs. region size); they call these selected regions the "potentially feasible" regions. Second, they combine these potentially feasible regions with the regions that are strictly feasible (no constraint violation) and apply the normal selection process (based on region function value vs. region size). They call the resulting constrained algorithm eDIRECT-C. Similarly to eDIRECT, meta-models are fit to the objective and constraints, and the optimum on the surfaces is computed and sampled.

Costa et al. [7] introduce a constrained version of DIRECT that also handles constraints via a modification of rectangle selection. Motivated by the filter methodology of Fletcher and Leyffer [16], they identify infeasible rectangles that are non-dominated with respect to the sum of constraint violations and centerpoint function value. Rectangle selection is then done separately for the rectangles that are feasible, those that are infeasible and nondominated, and those that are infeasible and dominated. For feasible rectangles, the selection is done as usual by finding the lower-right convex hull in a plot of centerpoint function value versus rectangle size; for infeasible rectangles, it is done based on a plot of the sum constraint violations versus rectangle size. The key contribution is showing that considering the nondominated and dominated infeasible rectangles separately gives better results than combining them.

Di Pillo et al. [11] present a two-level selection process to solve constrained problems. First, they identify rectangles that are potentially optimal with respect to total constraint violation and rectangle size. Second, of the rectangles selected in the first step, they select those that are potentially optimal with respect to function value and rectangle size. This procedure is very similar to that used in eDIRECT-C. Similarly to DIRMIN, they also explore launching a derivative-free constrained local optimizer from the center of each potentially optimal rectangle.

Stripinis et al. [69] extended their DIRECT-GL method described earlier to handle nonlinear constraints, creating what they call DIRECT-GLc. If a feasible point has not yet been found, they do iterations of DIRECT-GL to minimize the sum of constraint violations. Once a feasible point has been found, they apply DIRECT-GL to minimize the objective function plus a special penalty. For feasible points, this penalty is zero. For infeasible points, the penalty is the sum of constraint violations plus $|f(x) - f_{\min}|$, where $f_{\min}$ is the objective value of the best feasible point found so far. This penalty does not allow rectangles that achieve $f(x) < f_{\min}$ by violating the constraints to "get credit" for having a value of the objective function below $f_{\min}$. The advantage of this special penalty is that it works without the need to compute/adjust penalty weights on the constraint violations. The disadvantage of the penalty function is that it is discontinuous at the boundary of the feasible region (where optima often are located), and DIRECT is not guaranteed to converge where the function is discontinuous. To manage this potential problem, the authors introduce a modified version called DIRECT-GLce that does not impose the penalty if the constraint violations are below some tolerance $\epsilon_{cons}$ that is adaptively updated over the course of the search. The authors have also hybridized this method with a local optimizer to get a method they call DIRECT-GLce-min, and find that doing so drastically speeds up convergence on the test problems they consider, often by an order of magnitude.

Di Pillo et al. [10] introduce DF-EPGO, a constrained version of DIRECT based on using an exact penalty function. In each iteration, the penalized objective function is optimized with DIRECT and then the penalty parameters are updated. After a finite number of iterations, the procedure is guaranteed to converge.

In addition to the above extensions to handle nonlinear constraints, DIRECT has been extended in other ways. Multi-objective versions of DIRECT have been proposed, with a

review by Wong et al. [75] and a recent idea by Lovison and Miettinen [43] that is still under development. Problems with noise in the objective function present a special challenge (e.g., the value of $f_{\min}$ is only known approximately due to noise) and have been considered by Deng and Ferris [9]. Extensions to handle the case when the objective function is evaluated by a simulation that can fail (and thus no value is provided) have been proposed by Carter et al. [5] as well as Na et al. [48]. Adaptations of DIRECT for symmetric problems—problems in which the objective is the same for any permutation of the inputs—have been proposed by Grbić et al. [21] and by Paulavičius and Žilinskas [50]. Tao et al. [72] explore addressing higher-dimensional problems by using DIRECT to optimize over randomly-chosen subsets of one or two variables at a time (block coordinate descent). Finally, Scitovski and Sabo [59] use DIRECT to help solve partitioning problems: they run DIRECT for few iterations to get a good initial partition, and then fine-tune with the $k$-means clustering algorithm.

## 4 Conclusion

In this paper we reviewed the original DIRECT algorithm, highlighted its strengths and weaknesses, and surveyed the efforts by various authors to modify and extend DIRECT to address these weaknesses. Each modification has unique features, but we can roughly divide them into five groups, based on their key innovation, as shown in Table 1.

The first group keeps DIRECT's rectangular partitioning strategy, but modifies how rectangles are selected for further search. This group includes the locally biased DIRECT (DIRECT-l) [18]; multi-level DIRECT (MrDIRECT) [38,39]; DIRECT-GL [68]; and ADC [63]. All these methods focus on addressing the first two weaknesses of DIRECT: low convergence rate in solution refinement and delay in finding the global optimum due to excessive local search around a local minimum or insufficient global search. These methods have the advantage that they stay within the DIRECT framework of a rectangular space partition and avoid hybridizing DIRECT with a separate local optimizer.

The second group includes methods that explore modifications to DIRECT's partitioning scheme. The papers by Paulavičius et al. [50,51] explore the possible advantages of partitioning the space into simplexes as opposed to rectangles. The conclusion was that sampling the vertices of the simplexes was better than sampling their centroids, and that this approach may have a small advantage over rectangular partitions. Intuitively, the advantage of simplex partitioning is that each region is represented by $n + 1$ sampled points as opposed to just one and, hence, the true value of sampling in the region may be better captured. On the other hand, because this approach starts by sampling all $2^n$ corners of the search space, it is limited in practice to low-dimensional problems. The papers by Paulavičius et al. [52,53] consider rectangular partitions, but subdivide rectangles using bisection instead of trisection and sample two points within each rectangle instead of one.

The third group only contains HD-DIRECT. This method addresses the fact that DIRECT performs poorly in high dimensions by running successive cycles of DIRECT using ever more restrictive lower and upper bounds. As we have seen, this definitely speeds up convergence relative to DIRECT on high-dimensional problems, especially when high accuracy is desired. But the algorithm is still extremely slow on simple convex problems such as the sphere problem (minimize $\sum_1^{15} x_k^2$ on $[-3, 7]^{15}$). Like DIRECT itself, HD-DIRECT has no mechanism to extrapolate/interpolate clear trends in the data.

The fourth group hybridizes DIRECT with a separate local optimization algorithm. In our view, this is a key innovation: the local optimizer can rapidly fine tune a local solution found

**Table 1** Categorization of the surveyed modifications to DIRECT

| Key modification | References | Algorithm |
|---|---|---|
| Rectangle selection | Gablonsky and Kelly [18] Finkel and Kelley [14] | Locally biased DIRECT (DIRECT-l ) |
| | Sergeyev and Kvasov [63] | Efficient diagonal sampling and two-phase selection (ADC) |
| | Liu et al. [38,39] | MrDIRECT and $\text{MrDIRECT}_{075}$ |
| | Stripinis et al. [68] | Two-step global-local selection (DIRECT-GL) |
| Partitioning method | Paulavicius et al. [50,51] | DISIMPL-C, DISIMPL-V, Gb-DISIMPL-V |
| | Paulavicius et al. [52,53] | BIRECT, Gb-BIRECT |
| Zoom-in strategy | Tavassoli et al. [73] | High-Dimensional DIRECT (HD-DIRECT) |
| Add local search | Liuzzi et al. [40] | DIRMIN |
| | Jones [29] | Revision of DIRECT |
| | Holmström [26] | glcCluster |
| | Paulavičius et al. [53] | Gb-BIRMIN |
| Flexible partition and metasurface | Liu et al. [36,37] | Voronoi-based partitioning with metasurface (eDIRECT) |

by DIRECT to higher accuracy, exploit any obvious local trends such as monotonicity, and improve performance in higher dimensions. Of the algorithms in this group, DIRMIN [40] is the most brute-force, running a local search from the center of every selected rectangle. Jones' revision of DIRECT [29] is more economical, running a local after the first 100 or so evaluations, and then again whenever continued global search with DIRECT improves upon the best local solution. The glcCluster method [26] runs DIRECT for quite a few function evaluations (the default is $100n+1$), clusters the sampled points to identify promising regions, and finally launches a local optimizer from the best point in each cluster. The last method in this group, Gb-BIRMIN [53], uses the globally-biased selection method to help find the location of the global minimum and hybridizes this with a local optimizer to refine solutions to high accuracy.

The fifth group uses a flexible partition strategy together with metasurfaces (e.g., kriging) to help interpolate and extrapolate trends in the data. The eDIRECT algorithm [36] has the most flexible partitioning strategy because it is based on Voronoi cells and can accept any set of initial points, such as a Latin hypercube. A kriging metamodel is fit to the observed function values at the sampled points and is used to help decide where to sample inside the potentially optimal Voronoi cells. The kriging model is also optimized over the entire space (thereby exploiting any clear trends in the data), and the solution is added to the Voronoi partition.

Which of these extensions is best? This question is hard to answer because there are few common problems for which results are compared. Moreover, which algorithm works best depends on the difficulty and dimensionality of the problem. However, there are some features that we conclude to be generally beneficial:

- Hybridizing DIRECT with a local optimizer can be a key enabler for speeding up convergence.
- Splitting a selected rectangle on *one* long side, instead of all long sides, generally speeds up convergence.
- When several rectangles are tied for being potentially optimal, selecting just one (instead of all of them) speeds up convergence.
- Detecting and exploiting trends by fitting global or local response surface models to the sampled points makes more thorough use the data and can improve performance. Clustering the sampled points to identify promising areas for local search (as in glcCluster) is another way to exploit trends in the sampled data.
- Adding a mechanism, such as DIRECT's $\epsilon$ parameter, to stop the selection and subdivision of extremely small rectangles is generally beneficial, as it avoids wasting precious function evaluations on very small improvements.
- Sampling more than one point per partition region, such as in BIRECT [52] or DISIMPL-V [51], makes it less likely that DIRECT converges slowly because the region with the global minimum has been "unluckily" sampled at a nearby point with a bad function value.

A special challenge for all these methods is working effectively across problems of varying levels of difficulty. Most of the methods have a fixed strategy for how they balance local and global search and do not adapt based on the results of the evaluations. For example, the DIRECT-GL method does a global and local selection step each iteration, regardless of whether or not more local search is actually needed (e.g., if the current best solution has remained unchanged for several iterations, one might conclude that there is no need for another local step). The ADC and Gb-BIRECT methods are exceptions in that they continue the local search phase until no improvement is made, and only then switch to the global

search phase. Similarly, the global phase is continued until it improves upon the last local solution, at which point the local phase is started.

Additional research opportunities exist in mixing and matching features from the different algorithms surveyed, trying to combine multiple good features in one package. For example, the efficient diagonal sampling strategy of the ADC method, or the bisection method of BIRECT, could be substituted for centerpoint sampling in the original DIRECT, DIRECT-GL, or MrDIRECT algorithms. Borrowing an idea from eDIRECT, a kriging response surface could be periodically fit to the points sampled by the other DIRECT variants and the surface then optimized to suggest a good point for sampling or for starting a local search. The adaptive diagonal curves (ADC) method could be hybridized with a local optimizer to enable it to be competitive not only on hard problems when we have a large computational budget, but also on smaller problems with a lower computational budget. The zoom-in strategy of HD-DIRECT could be combined with the other DIRECT variants to help those variants stretch to higher dimensions. The DIRECT-GL and PLO methods could keep their special selection method (selecting rectangles nondominated on function value and size), but borrow DIRECT's method using the $\epsilon$ parameter to put a limit on how small a selected rectangle can be. These are just some of many possibilities. If the past is any precedent, we can expect more DIRECT variants to appear in the next 25 years.

**Acknowledgements** K. Ljungberg, S. Holmgren, and Ö Carlborg kindly granted us permission to reproduce graphics from [42] in our Figure 9. Q. Liu, G. Yang, Z. Zhang, and J. Zeng kindly granted us permission to reproduce graphics from [39] in our Figure 23. H. Liu, S. Xu, X. Chen, X. Wang, and Q. Ma kindly granted us permission to reproduce graphics from [36] in our Figures 29 and 30. Shugo Kaneko suggested ways to improve the clarity of the discussion around Figure 30 as well as our description of the HD-DIRECT algorithm. Finally, we would like to convey deep thanks to the two reviewers who read this long paper very carefully, suggesting many possible improvements and bringing to our attention several key articles we had overlooked.



## References

1. Bartholomew-Biggs, M.C., Parkhurst, S.C., Wilson, S.P.: Using DIRECT to solve an aircraft routing problem. Comput. Optim. Appl. **21**(3), 311–323 (2002). https://doi.org/10.1023/A:1013729320435
2. Belfkira, R., Zhang, L., Barakat, G.: Optimal sizing study of hybrid wind/PV/diesel power generation unit. Solar Energy **85**(1), 100–110 (2011). https://doi.org/10.1016/j.solener.2010.10.018
3. Campana, E.F., Diez, M., Iemma, U., Liuzzi, G., Lucidi, S., Rinaldi, F., Serani, A.: Derivative-free global ship design optimization using global/local hybridization of the DIRECT algorithm. Optim. Eng. **17**(1), 127–156 (2016). https://doi.org/10.1007/s11081-015-9303-0
4. Cappellari, M., Verolme, E.K., van der Marel, R.P., Kleijn, G.A.V., Illingworth, G.D., Franx, M., Carollo, C.M., deZeeuw, P.T.: The counterrotating core and the black hole mass of IC 1459. Astrophys. J. **578**(2), 787–805 (2002). https://doi.org/10.1086/342653
5. Carter, R.G., Gablonsky, J.M., Patrick, A., Kelley, C.T., Eslinger, O.J.: Algorithms for noisy problems in gas transmission pipeline optimization. Optim. Eng. **2**(2), 139–157 (2001). https://doi.org/10.1023/A:1013123110266
6. Chang, Y., Hung, K., Lee, S.: Human face detection with neural networks and the DIRECT algorithm. Artif. Life Robot. **12**(1), 112 (2008). https://doi.org/10.1007/s10015-007-0491-3

7. Costa, M.F.P., Rocha, A.M.A., Fernandes, E.M.: Filter-based DIRECT method for constrained global optimization. J. Glob. Optim. **71**(3), 517–536 (2018). https://doi.org/10.1007/s10898-017-0596-8
8. Cox, S.E., Haftka, R.T., Baker, C.A., Grossman, B., Mason, W.H., Watson, L.T.: A comparison of global optimization methods for the design of a high-speed civil transport. J. Glob. Optim. **21**(4), 415–432 (2001). https://doi.org/10.1023/A:1012782825166
9. Deng, G., Ferris, M.C.: Extension of the DIRECT optimization algorithm for noisy functions. In:Proceedings of the 39th Conference on Winter Simulation: 40 Years! The Best is Yet to Come (WSC'07), pp. 497–504. IEEE Press, Piscataway (2007). http://dl.acm.org/citation.cfm?id=1351542.1351641
10. Di Pillo, G., Lucidi, S., Rinaldi, F.: A derivative-free algorithm for constrained global optimization based on exact penalty functions. J. Optim. Theory Appl. **164**(3), 862–882 (2015). https://doi.org/10.1007/s10957-013-0487-1
11. Di Pillo, G., Liuzzi, G., Lucidi, S., Piccialli, V., Rinaldi, F.: A DIRECT-type approach for derivative-free constrained global optimization. Comput. Optim. Appl. **65**(2), 361–397 (2016). https://doi.org/10.1007/s10589-016-9876-3
12. Elsakov, S.M., Shiryaev, V.I.: Homogeneous algorithms for multiextremal optimization. Comput. Math. Math. Phys. **50**(10), 1642–1654 (2010). https://doi.org/10.1134/S0965542510100027
13. Fellini, R., Michelena, N., Papalambros, P., Sasena, M.: Optimal design of automotive hybrid powertrain systems. In: Proceedings First International Symposium on Environmentally Conscious Design and Inverse Manufacturing, pp. 400–405 (1999). https://doi.org/10.1109/ECODIM.1999.747645
14. Finkel, D.E., Kelley, C.T.: Additive scaling and the DIRECT algorithm. J. Glob. Optim. **36**(4), 597–608 (2006). https://doi.org/10.1007/s10898-006-9029-9
15. Finkel, D.E., Kelly, C.T.: An adaptive restart implementation of DIRECT. Technical report, Technical report CRSC-TR04-30, Center for Research in Scientific Computation, North Carolina State University, Raleigh (2004). https://repository.lib.ncsu.edu/bitstream/handle/1840.4/461/crsc-tr04-30.pdf?sequence=1
16. Fletcher, R., Leyffer, S.: Nonlinear programming without a penalty function. Math. Program. **269**, 239–269 (2002). https://doi.org/10.1007/s101070100244
17. Gablonsky, J.: Modifications of the DIRECT Algorithm. PhD thesis, North Carolina State University, Raleigh (2001). https://repository.lib.ncsu.edu/handle/1840.16/3920
18. Gablonsky, J., Kelly, C.T.: A locally-biased form of the DIRECT algorithm. J. Glob. Optim. **21**(1), 27–37 (2001). https://doi.org/10.1023/A:1017930332101
19. Gaviano, M., Kvasov, D.E., Lera, D., Sergeyev, Y.D.: Algorithm 829: software for generation of classes of test functions with known local and global minima for global optimization. ACM Trans. Math. Softw. **29**(4), 469–480 (2003). https://doi.org/10.1145/962437.962444
20. Gillard, J.W., Kvasov, D.E.: Lipschitz optimization methods for fitting a sum of damped sinusoids to a series of observations. Stat. Its Interface **10**(1), 59–70 (2017). https://doi.org/10.4310/SII.2017.v10.n1.a6
21. Grbić, R., Nyarko, E.K., Scitovski, R.: A modification of the DIRECT method for Lipschitz global optimization for a symmetric function. J. Glob. Optim. **57**(4), 1193–1212 (2013). https://doi.org/10.1007/s10898-012-0020-3
22. Hansen, P., Jaumard, B.: Lipschitz optimization. In: Horst, R., Pardalos, P.M. (eds.) Handbook of Global Optimization, pp. 407–493. Springer, Boston (1995). https://doi.org/10.1007/978-1-4615-2025-2_9
23. Hansen, P., Jaumard, B., Lu, S.H.: On using estimates of Lipschitz constants in global optimization. J. Optim. Theory Appl. **75**(1), 195–200 (1992). https://doi.org/10.1007/BF00939912
24. Hao, J., Yu, Z., Zhao, Z., Shen, P., Zhan, X.: Optimization of key parameters of energy management strategy for hybrid electric vehicle using DIRECT algorithm. Energies **9**(12), 997 (2016). https://doi.org/10.3390/en9120997
25. Hedar, A.: Global Optimization Test Problems. http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm (2020)
26. Holmström, K.: TOMLAB GlcCluster. http://tomwiki.com/GlcCluster#Description/
27. Huyer, W., Neumaier, A.: Global optimization by multilevel coordinate search. J. Glob. Optim. **14**(4), 331–355 (1999). https://doi.org/10.1023/A:1008382309369
28. Jarvis, R.A.: On the identification of the convex hull of a finite set of points in the plane. Inf. Process. Lett. **2**(1), 18–21 (1973). https://doi.org/10.1016/0020-0190(73)90020-3
29. Jones, D.R.: Direct global optimization algorithm. In: Floudas, C.A., Pardalos, P.M. (Eds.) Encyclopedia of Optimization, pp. 431–440. Springer, Boston (2001). https://doi.org/10.1007/0-306-48332-7_93
30. Jones, D.R., Perttunen, C.D., Stuckman, B.E.: Lipschitzian optimization without the Lipschitz constant. J. Optim. Theory Appl. **79**(1), 157–181 (1993). https://doi.org/10.1007/BF00941892

31. Kokail, C., Maier, C., van Bijnen, R., Brydges, T., Joshi, M.K., Jurcevic, P., Muschik, C.A., Silvi, P., Blatt, R., Roos, C.F., Zoller, P.: Self-verifying variational quantum simulation of lattice models. Nature **569**(7756), 355–360 (2019). https://doi.org/10.1038/s41586-019-1177-4
32. Kvasov, D.E., Pizzuti, C., Sergeyev, Y.D.: Local tuning and partition strategies for diagonal GO methods. Numer. Math. **94**(1), 93–106 (2003). https://doi.org/10.1007/s00211-002-0419-8
33. Kvasov, D.E.: Multidimensional lipschitz global optimization based on efficient diagonal partitions. 4OR **6**, 403–406 (2008). https://doi.org/10.1007/s10288-007-0065-1
34. Kvasov, D.E., Menniti, D., Pinnarelli, A., Sergeyev, Y.D., Sorrentino, N.: Tuning fuzzy power-system stabilizers in multi-machine systems by global optimization algorithms based on efficient domain partitions. Electr. Power Syst. Res. **78**(7), 1217–1229 (2008). https://doi.org/10.1016/j.epsr.2007.10.009
35. Lang, H., Liu, L., Yang, Q.: Design of URAs by DIRECT global optimization algorithm. Optik **120**(8), 370–373 (2009). https://doi.org/10.1016/j.ijleo.2007.09.010
36. Liu, H., Xu, S., Wang, X., Wu, J., Song, Y.: A global optimization algorithm for simulation-based problems via the extended DIRECT scheme. Eng. Optim. **47**(11), 1441–1458 (2015). https://doi.org/10.1080/0305215X.2014.971777
37. Liu, H., Xu, S., Chen, X., Wang, X., Ma, Q.: Constrained global optimization via a DIRECT-type constraint-handling technique and an adaptive metamodeling strategy. Struct. Multidiscip. Optim. **55**(1), 155–177 (2017). https://doi.org/10.1007/s00158-016-1482-6
38. Liu, Q., Zeng, J., Yang, G.: MrDIRECT: a multilevel robust DIRECT algorithm for global optimization problems. J. Glob. Optim. **62**(2), 205–227 (2015). https://doi.org/10.1007/s10898-014-0241-8
39. Liu, Q., Yang, G., Zhang, Z., Zeng, J.: Improving the convergence rate of the DIRECT global optimization algorithm. J. Glob. Optim. **67**(4), 851–872 (2017). https://doi.org/10.1007/s10898-016-0447-z
40. Liuzzi, G., Lucidi, S., Piccialli, V.: A DIRECT-based approach exploiting local minimizations for the solution of large-scale global optimization problems. Comput. Optim. Appl. **45**(2), 353–375 (2010). https://doi.org/10.1007/s10589-008-9217-2
41. Liuzzi, G., Lucidi, S., Piccialli, V.: Exploiting derivative-free local searches in DIRECT-type algorithms for global optimization. Comput. Optim. Appl. **65**(2), 449–475 (2016). https://doi.org/10.1007/s10589-015-9741-9
42. Ljungberg, K., Holmgren, S., Carlborg, Ö.: Simultaneous search for multiple QTL using the global optimization algorithm DIRECT. Bioinformatics **20**(12), 1887–1895 (2004). https://doi.org/10.1093/bioinformatics/bth175
43. Lovison, A., Miettinen, K.: Exact extension of the DIRECT algorithm to multiple objectives. AIP Conf. Proc. **2070**(1), 020053 (2019). https://doi.org/10.1063/1.5090020
44. Marot, J., Bourennane, S.: Subspace-based and DIRECT algorithms for distorted circular contour estimation. IEEE Trans. Image Process. **16**(9), 2369–2378 (2007). https://doi.org/10.1109/TIP.2007.903907
45. Mockus, J.: On the pareto optimality in the context of Lipschitzian optimization. Informatica **22**(4), 521–536 (2011). https://doi.org/10.15388/INFORMATICA.2011.340
46. Mockus, J., Paulavičius, R., Rusakevičius, D., Šešok, D., Žilinskas, J.: Application of reduced-set Pareto-Lipschitzian optimization to truss optimization. J. Glob. Optim. **67**(1), 425–450 (2017). https://doi.org/10.1007/s10898-015-0364-6
47. Moles, C.G., Mendes, P., Banga, J.R.: Parameter estimation in biochemical pathways: a comparison of global optimization methods. Genome Res. **13**(11), 2467–2474 (2003). https://doi.org/10.1101/gr.1262503
48. Na, J., Lim, Y., Han, C.: A modified DIRECT algorithm for hidden constraints in an LNG process optimization. Energy **126**, 488–500 (2017). https://doi.org/10.1016/j.energy.2017.03.047
49. Nguyen, T.L., Low, K.: A global maximum power point tracking scheme employing DIRECT search algorithm for photovoltaic systems. IEEE Trans. Ind. Electron. **57**(10), 3456–3467 (2010). https://doi.org/10.1109/TIE.2009.2039450
50. Paulavičius, R., Žilinskas, J.: Simplicial Lipschitz optimization without Lipschitz constant. In: Simplicial Global Optimization, pp. 61–86. Springer, New York (2014) https://doi.org/10.1007/978-1-4614-9093-7_3
51. Paulavičius, R., Sergeyev, Y.D., Kvasov, D.E., Žilinskas, J.: Globally-biased DISIMPL algorithm for expensive global optimization. J. Glob. Optim. **59**(2), 545–567 (2014). https://doi.org/10.1007/s10898-014-0180-4
52. Paulavičius, R., Chiter, L., Žilinskas, J.: Global optimization based on bisection of rectangles, function values at diagonals, and a set of Lipschitz constants. J. Glob. Optim. **71**(1), 5–20 (2018). https://doi.org/10.1007/s10898-016-0485-6
53. Paulavičius, R., Sergeyev, Y.D., Kvasov, D.E., Žilinskas, J.: Globally-biased BIRECT algorithm with local accelerators for expensive global optimization. Expert Syst. Appl. **144**, 113052 (2020). https://doi.org/10.1016/j.eswa.2019.113052

54. Pintér, J.D.: Global optimization in action. In: Pardalos, P. (ed.) Nonconvex Optimization and its Applications, vol. 6. Springer, USA (1996). https://doi.org/10.1007/978-1-4757-2502-5
55. Piyavskii, S.A.: An algorithm for finding the absolute extremum of a function. USSR Comput. Math. Math. Phys. **12**(4), 57–67 (1972). https://doi.org/10.1016/0041-5553(72)90115-2
56. Powell, M.J.D.: The BOBYQA algorithm for bound constrained optimization without derivatives. Technical report, Department of Applied Mathematics and Theoretical Physics, University of Cambridge (2009). http://www.damtp.cam.ac.uk/user/na/NA_papers/NA2009_06.pdf
57. Rios, L.M., Sahinidis, N.V.: Derivative-free optimization: a review of algorithms and comparison of software implementations. J. Glob. Optim. **56**, 1247–1293 (2013). https://doi.org/10.1007/s10898-012-9951-y
58. Ruf, F., Neiss, A., Barthels, A., Kohler, T.P.,Michel, H., Froeschl, J., Herzog, H.: Design optimization of a 14 V| automotive power net using a parallelized DIRECT algorithm in a physical simulation. In: 2012 13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM), pp. 73–80 (2012). https://doi.org/10.1109/OPTIM.2012.6231911
59. Scitovski, Rudolf, Sabo, Kristian: Application of the DIRECT algorithm to searching for an optimal $k$-partition of the set $a \in r^n$ and its application to the multiple circle detection problem. J. Glob. Optim. **74**(1), 63–77 (2019). https://doi.org/10.1007/s10898-019-00743-8
60. Sergeyev, Y.D.: An information global optimization algorithm with local tuning. SIAM J. Optim. **5**(4), 858–870 (1995). https://doi.org/10.1137/0805041
61. Sergeyev, Y.D.: Global one-dimensional optimization using smooth auxiliary functions. Math. Program. **81**(1), 127–146 (1998). https://doi.org/10.1007/BF01584848
62. Sergeyev, Y.D.: Efficient strategy for adaptive partition of n-dimensional intervals in the framework of diagonal algorithms. J. Optim. Theory Appl. **107**(1), 145–168 (2000). https://doi.org/10.1023/A:1004613001755
63. Sergeyev, Y.D., Kvasov, D.E.: Global search based on efficient diagonal partitions and a set of Lipschitz constants. SIAM J. Optim. **16**(3), 910–937 (2006). https://doi.org/10.1137/040621132
64. Sergeyev, Y.D., Kvasov, D.E.: Deterministic Global Optimization: An Introduction to the Diagonal Approach. Springer Briefs in Optimization. Springer, Berlin (2017). https://doi.org/10.1007/978-1-4939-7199-2
65. Sergeyev, Y.D., Kvasov, D.E., Mukhametzhanov, M.S.: On the efficiency of nature-inspired metaheuristics in expensive global optimization with limited budget. Sci. Rep. **8**(1), 453 (2018). https://doi.org/10.1038/s41598-017-18940-4
66. Sergeyev, Y.D., Kvasov, D.E., Mukhametzhanov, M.S.: On strong homogeneity of a class of global optimization algorithms working with infinite and infinitesimal scales. Commun. Nonlinear Sci. Numer. Simul. **59**, 319–330 (2018). https://doi.org/10.1016/j.cnsns.2017.11.013
67. Shubert, B.: A sequential method seeking the global maximum of a function. SIAM J. Numer. Anal. **9**(3), 379–388 (1972). https://doi.org/10.1137/0709036
68. Stripinis, L., Paulavičius, R., Žilinskas, J.: Improved scheme for selection of potentially optimal hyperrectangles in DIRECT. Optim. Lett. **12**(7), 1699–1712 (2018). https://doi.org/10.1007/s11590-017-1228-4
69. Stripinis, L., Paulavičius, R., Žilinskas, J.: Penalty functions and two-step selection procedure based DIRECT-type algorithm for constrained global optimization. Struct. Multidiscip. Optim. **59**(6), 2155–2175 (2019). https://doi.org/10.1007/s00158-018-2181-2
70. Strongin, R.G., Sergeyev, Y.D.: Global Optimization with Non-convex Constraints: Sequential and Parallel Algorithms, Volume 45 of Nonconvex Optimization and Its Applications. Springer, US (2000). https://doi.org/10.1007/978-1-4615-4677-1
71. Svensson, B., Nia, N.K., Danielsson, F., Lennartson, B.: Sheet-metal press line parameter tuning using a combined DIRECT and Nelder–Mead algorithm. In: ETFA2011, pp. 1–8 (2011). https://doi.org/10.1109/ETFA.2011.6059031
72. Tao, Q., Huang, X., Wang, S., Li, L.: Adaptive block coordinate DIRECT algorithm. J. Glob. Optim. **69**(4), 797–822 (2017). https://doi.org/10.1007/s10898-017-0541-x
73. Tavassoli, A., Hajikolaei, K.H., Sadeqi, S., Wang, G.G., Kjeang, E.: Modification of DIRECT for high-dimensional design problems. Eng. Optim. **46**(6), 810–823 (2014). https://doi.org/10.1080/0305215X.2013.800057
74. Wipke, K., Markel, T., Nelson, D.: Optimizing energy management strategy and a degree of hybridization for a hydrogen fuel cell SUV. In: Proceedings of the 18th Electric Vehicle Symposium(EVS-18), Berlin, Germany (2001). https://pdfs.semanticscholar.org/b1cd/d6b0ab88dd50b2d228854bd9de3512785444.pdf
75. Wong, C.S.Y., Al-Dujaili, A., Sundaram, S.: Hypervolume-based DIRECT for multi-objective optimisation. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion,

GECCO'16 Companion, pp. 1201–1208 (2016). ACM, New York. https://doi.org/10.1145/2908961.2931702
76. Xiao, Yiming, Rivaz, Hassan, Chabanas, Matthieu, Fortin, Maryse, Machado, Ines, Yangming, Ou, Heinrich, Mattias P., Schnabel, Julia A., Zhong, Xia, Maier, Andreas, et al.: Evaluation of MRI to ultrasound registration methods for brain shift correction: the CuRIOUS2018 challenge. IEEE Trans. Med. Imaging **39**(3), 777–786 (2020). https://doi.org/10.1109/TMI.2019.2935060
77. Zhu, H., Bogy, D.B.: DIRECT algorithm and its application to slider air-bearing surface optimization. IEEE Trans. Magn. **38**(5), 2168–2170 (2002). https://doi.org/10.1109/TMAG.2002.802794
78. Žilinskas, A.: On strong homogeneity of two global optimization algorithms based on statistical models of multimodal objective functions. Appl. Math. Comput. **218**(16), 8131–8136 (2012). https://doi.org/10.1016/j.amc.2011.07.051. Special Issue dedicated to the international workshop 'Infinite and Infinitesimal in Mathematics, Computing and Natural Sciences'